INCREMENTAL SENTENCE GENERATION

a computer model of grammatical encoding

Koenraad J.M.J. De Smedt

INCREMENTAL SENTENCE GENERATION

A COMPUTER MODEL OF GRAMMATICAL ENCODING

Een wetenschappelijke proeve op het gebied van de Sociale Wetenschappen,
in het bijzonder de Psychologie

Proefschrift

ter verkrijging van de graad van Doctor aan de Katholieke Universiteit te Nijmegen,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op dinsdag 27 maart 1990
des namiddags te 3:30 uur

door

KOENRAAD JAN MARIA JOHANNA DE SMEDT

geboren op 30 oktober 1954 te Wilrijk (België)

NICI Technical Report 90-01

Nijmegen Institute for Cognition Research and Information Technology

Promotores: Prof. Dr. G.A.M. Kempen
Prof. Dr. L. Steels (V.U.Brussel)

ISBN 90 373 0049 9 / CIP / NUGI 941

Printed in the Netherlands by Samsom-Sijthoff grafische bedrijven bv

Preface

Computational psycholinguistics is a scientific field which is concerned with the way in which intelligent systems perform linguistic tasks under realistic psychological constraints. Language understanding and generation are the most fundamental of these tasks. This study focuses on generation, and it does so by presenting a computer simulation program.

The language generation task has long been studied experimentally. We now know, e.g, that human speakers perform this task under certain constraints of time and memory, that they make certain errors, that certains aspects of generation are more vulnerable than others (in aphasia), etc. If on the basis of these observations we want to develop a theory of *what happens*, we need a formalism to talk about processes. The computer metaphor encourages us to talk about processes in terms of *programs*. By means of computer programs, psycholinguistic processes can be simulated with a precision and explicitness as never before. Even if it is not possible to prove, by constructing a computer program, that a theory is adequate or valid, it may be possible to discover flaws in the theory by analyzing the program and comparing its behavior with experimental data. In addition, the computer also proves to be a practical tool: simulations help to organize knowledge and visualize complex cognitive processes.

The work reported in this dissertation is a meeting point of two distinct lines of thought. One is *Incremental Procedural Grammar* (IPG), a theory of human language generation proposed by Kempen and Hoenkamp (1987). The second is the representation of linguistic knowledge in an *object-oriented* programming paradigm. Both lines meet in a new computer model of incremental sentence generation where linguistic concepts are represented in an object-oriented way.

A *procedural* grammar, according to Kempen and Hoenkamp, is a theory of language "which articulates—preferably empirically grounded—assumptions about *both* format of grammar rules *and* structure and functioning of the syntactic processor" (1987:209). An *incremental* grammar is a theory of language which accounts for the piecemeal generation of sentences. Kempen and Hoenkamp have pushed the procedural view to the extreme by proposing a grammar which is identical to the structure of a language processor. The syntactic structure of a sentence is represented *implicitly* in the hierarchy of subprocedures in the generation process; calling relations are grammatical relations. This compact and elegant computer model opens up unusual possibilities for linguistic representation, e.g., modeling coordination as iteration, and manipulating the scope of variables to account for *clause union*. However, since a procedure call hierarchy is directed from the top downward, the syntactic structure must also be built

in a top-down fashion. This restriction is unmotivated from the viewpoint of incremental generation. Moreover, the lack of a declarative representation and the coalescence of control knowledge and domain knowledge are obstacles for the expression of linguistic generalizations.

An object-oriented representation of knowledge captures just these generalizations. The expression of linguistic knowledge as a network of entities (*objects*) related by propositions is the foundation of an efficient and extensible knowledge base. A prime relation between objects is *default inheritance*: objects can share their common behavior by letting one object *inherit* knowledge from another one. If linguistic categories are represented as objects, regularities in their behavior can be modeled by means of inheritance. The use of inheritance avoids redundancy and allows the handling of exceptions without disturbing general knowledge. CommonORBIT is a simple but powerful object-oriented programming language which can serve as a concrete framework for linguistic representation.

The catalysis between both lines of research was effectuated by the development of *Segment Grammar* (SG; Kempen, 1987). This new grammar formalism proved especially suitable for incremental sentence generation because it allows the piecemeal construction of syntactic structures. SG serves as a grammar formalism in *IPF* (Incremental Parallel Formulator), a computer simulation program for sentence formulation. IPF not only enhances SG by fully representing the formalism in an object-oriented language, but also adds independent programming assumptions: the formulation task is *distributed* over a number of parallel processes. Clearly, this new machinery marks a departure from the original IPG theory. Still, many of the original psycholinguistic assumptions concerning language structures and their generation are maintained. E.g., the conceptual and lexical guidance of the generation process and the distinct stages in the generation process are never contested.

Chapter 1 of the dissertation presents the main problem. The reader is introduced to incremental sentence generation from the linguistic and psycholinguistic viewpoints. The human language processing apparatus is divided in distinct modules which are each responsible for a specialized task: a Conceptualizer, a Formulator, and an Articulator. It is then shown how a piecemeal flow of information between modules affects the linguistic form of the utterances. In a brief overview of IPF, the main claims of this thesis are expounded.

The remainder of this document is divided into two parts. Part One (Chapters 2 through 5) is concerned with psychological and linguistic aspects. The SG formalism and the IPF computer model are presented, with an emphasis on the structure and processing of language. Part Two (Chapters 6 through 9) is concerned with questions of representation and computation. It elaborates several issues raised in Part One from the perspective of AI programming.

Chapter 10 concludes with a brief evaluation of IPF/SG and a description of a possible extension of the model.

Acknowledgements

This research was started at the University of Antwerpen (UIA) with financial support of the Belgian National Foundation for Scientific Research (NFWO). It was continued at the University of Nijmegen, where it was partly sustained by grants from the Dutch Ministries of Education and Science and of Economic Affairs. I am grateful to the University of Nijmegen, who allowed me to spend time on the greater part of this work. I am also indebted to Schlumberger-Doll Research for their hospitality during two short visits.

More than to anyone else, I am indebted to Gerard Kempen, who as my main supervisor provided inspiration and stimulating enthusiasm. He took the role of a verbal sparring partner in the many discussions we had during the preparation of this thesis, and much of the credit for this work belongs to him.

The prehistory of my interest in computational models of language processing was dominated by Luc Steels, whose classes on computational linguistics at the University of Antwerpen were eye-openers. Throughout the many years that I have known him, his visionary creativity provided a welcome antithesis to my own hair-splitting.

The members of my manuscript committee, Willem Levelt, Eduard Hoenkamp and Walter Daelemans, provided instrumental comments on the preliminary version of this dissertation.

Furthermore, I acknowledge all the people who contributed to this work in a less direct, but no less important way, through discussions and friendly advice. This holds especially for all my past and present colleagues at the Language section of NICI.

Among the many teachers who introduced me to the fascinating world of linguistics, three should be singled out: Louis Goossens, Georges De Schutter and Frans Daems. Their classes were responsible for my complete addiction to the field.

Last but not least, very special thanks go to my parents and my wife Ria, who have always stimulated me to pursue my scientific interests and offered their support in more ways than one.

1 Introduction[1]

Natural speech is often produced in a piecemeal fashion: speakers start to articulate a sentence before the syntactic structure, or even the meaning content of that sentence has been fully determined. Under the assumption that the human language processing apparatus is capable of carrying out different tasks in parallel, the speaker may already utter the first words while simultaneously processing more content to be incorporated in the sentence. If the next fragment is ready to be uttered when the first fragment nears completion, speech is produced fluently and without hesitation. This mode of generation, which is called *incremental* generation, seems to serve a system whose major purpose is to articulate relatively fluent speech, even if it is imperfect or incomplete. Once a partial sentence has been constructed, the generator will try to complete the sentence in a maximally grammatical way. However, the integration of new content in an existing partial sentence is not always possible, and a repair or restart may be necessary. This chapter explores the global effects which an incremental mode of sentence generation exerts upon the form of the resulting utterances. After that, an overview is given of IPF (Incremental Parallel Formulator), a computer simulation model of grammatical encoding.

1.1 The language generation task

Speakers usually produce utterances in a spontaneous and seemingly effortless fashion. Yet the simulation of language generation on a computer meets many obstacles. The main difficulties for a computational model of the speaker can be attributed to two characteristics of the language generation task. First, language generation is a knowledge-intensive task. Although human speakers are usually not aware of it, there is an enormous amount of knowledge involved in the construction of an utterance. Natural languages are governed by a syntax which is extremely complicated, and a semantics which touches upon every human experience. The effective development of a language processing model therefore depends on an efficient and flexible organization of the knowledge involved.

Second, speaking is also a task which is critically affected by timing. It seems that the generation of an utterance has to take place within a certain time span due to limitations of memory and processing, and that various aspects of the generation task

[1] An earlier version of Sections 1.1, 1.2 and 1.4 was published as (De Smedt & Kempen, 1987).

are interleaved in the time dimension. If we want to construct a psychologically plausible sentence generator, we will have to replicate these conditions. While the present study does not pursue the goal of replication to the extreme, the passing of time during generation is nevertheless incorporated in the model as an important factor affecting the shape of an utterance. Let us now see how we can divide the generation process into several substages which can be plotted against the time dimension.

1.1.1 Stages of processing

Since Garrett's (1975, 1980) seminal work on speech error phenomena, it has become customary to distinguish several levels of representation within the sentence generation process: a message level, a functional level, a positional level, and a phonetic level (see also Bock, 1987). Garrett's model has been further elaborated and modified by Kempen (Kempen & Hoenkamp, 1987; Van Wijk & Kempen, 1987) who proposes the global sentence generation model depicted in Figure 1.1.

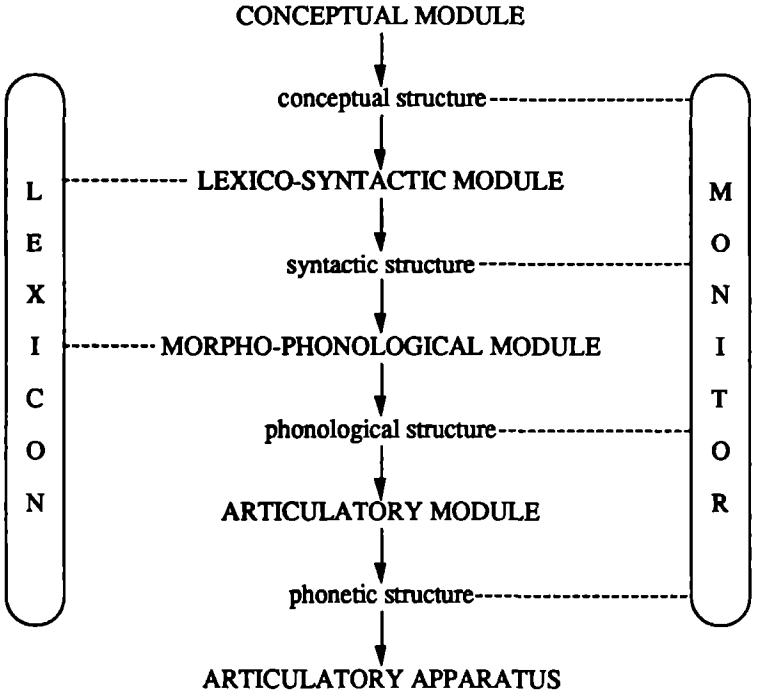


Figure 1.1
A global model of the sentence generation process

The four modules listed have the following functions:

1 The *conceptual* module forms a conceptual (semantic) representation of the message which the speaker wishes to communicate. The nature of the semantic structures which are output by this component need not concern us here.

2 The *lexico-syntactic* module constructs an ordered tree structure consisting of constituents and their functional relations. The terminal nodes of syntactic trees (both content and function words) are instances of syntactically specified lexical items called *lemmas* which are retrieved from the lexicon (cf. Kempen & Huijbers, 1983). While Garrett assigns the tasks of inserting function words and computing word order to a subsequent module (the positional stage), Kempen assigns them to this one.

3 The *morpho-phonological* module computes the word form of all lemmas by retrieving their phonological specifications (*lexemes*) from the lexicon and making various morphological and phonological adjustments.

4 The *articulatory* module produces a phonetic specification which is used to control the articulatory apparatus.

The intermediate results, which are passed from one module to another, are inspected by a *monitor* (Hoenkamp, 1980; Levelt, 1989). If the monitor notices that the output of one of the modules is inappropriate, or if it detects a violation of some prevailing constraint, any ongoing activity may be interrupted and backtracking to an earlier point in the generation process may be forced. This course of events may give rise to self-corrections.

Table 1.1
Components of the natural language generation process

A. CONCEPTUALIZER (What to say?)
A.1. Macro-planner (communicative intentions → illocutionary acts) • selecting information (discourse plans, reference, rhetorical structure) • grouping and ordering information (scripts, process constraints)
A.2. Micro-planner (illocutionary acts → preverbal messages) • reference (accessibility, conceptual prominence, topicalization) • assigning propositional format and perspective
B. FORMULATOR (How to say it?)
B.1. Grammatical Encoder (preverbal messages → surface structures) • lexicalization (lemma selection) • formation of syntactic structures (functional and surface structures)
B.2. Phonological Encoder (surface structures → phonetic plans) • selection of lexical forms (internal composition of words) • prosodic planning (intonation contour, metrical structure)
C. ARTICULATOR (Say it!)

In recent years, more attention has been paid to the conceptual and textual aspects of natural language generation; distinct substages are now recognized within the conceptual module. Table 1.1 presents an overall structure of the language generation process based on Levelt (1989), which is an elaboration of Kempen's model. Important tasks are listed for each stage. In Levelt's terminology, which will be frequently used in the remainder of the present work, the *Grammatical Encoder* corresponds to the lexico-syntactic module mentioned above while the *Phonological Encoder* corresponds to the morpho-phonological module. The output from the Formulator is directed to the *Articulator*, which is basically a motor component.

1.1.2 Incremental generation

The sequential modules of Figure 1.1 and Table 1.1 need not necessarily operate on input structures which correspond to whole sentences. If the modules did operate in this fashion, hesitations *during* the pronunciation of a sentence could not have a non-articulatory (i.e. a conceptual, syntactic, or lexical) origin. Also, it would not be possible for speakers to start articulating even very long sentences before having planned such sentences in detail at all levels. Since this is both counter-intuitive and counter-factual, I prefer the view that the modules can work on different parts of the final utterance simultaneously, forming a cascade schema depicted in Figure 1.2. I call this piecemeal mode of generation *incremental generation* (Kempen, 1978).

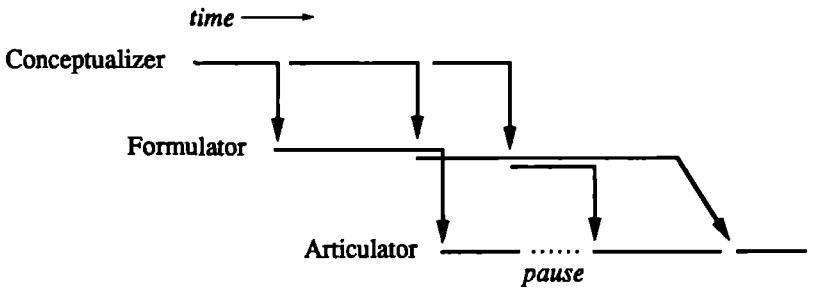


Figure 1.2
The cascade model of incremental generation

Although the modules involved in sentence generation work in parallel, each individual fragment of an utterance still goes through the different stages sequentially. An incremental framework can thus easily accommodate the fact that hesitations may occur within the sentence as well as between sentences. Also, it can account for *syntactic dead ends*, i.e., the fact that people sometimes 'talk themselves into a corner' when lexico-syntactic restrictions prevent a speaker from fitting new content into a partial, incomplete sentence. In such circumstances, self-corrections may be triggered. Moreover, the framework allows for 'changes of mind', i.e., decisions by the speaker to revise some conceptual content which has already been expressed. This is represented by marking a conceptual fragment as a substitute for an earlier fragment. Last but not least, an incremental architecture can account for certain variations in the output of the

Grammatical Encoder (in particular, word order and lexical choices) on the basis of timing variations in conceptual input.

Incremental sentence generation and self-correction will now be discussed from the point of view of their origins: conceptual modifications and monitoring. Then some grammatical encoding mechanisms for dealing with these events will be proposed.

1.1.3 Causes of incrementation and correction

Three basic kinds of modification to a conceptual structure will affect the shape of an utterance: *deletion*, *replacement* and *addition* of conceptual elements. Deletion and replacement will both give rise to a self-correction, which is often signaled by a pause or a correction term such as *uh*, *no*, or *sorry*. Some examples of deletion are (1a,b). Examples of replacement are (2a,b).

(1) a. John and Mary ...uh... only John went to a party last week.

b. John bought a new bicycle for ...uh... a bicycle for his son.

(2) a. John ...uh sorry... Mary went to the party.

b. The runner with the beard ...no... with the glasses is leading now.

Conceptual replacement may also lead to a non-retracing repair. The result is ungrammatical but contains no correction marker and is uttered without hesitation. The examples for English (3a) and for Dutch (3b,c) show how a constituent can be replaced without retracing. One or more constituents which have already been uttered are used as a hook to attach a new sentence pattern with a different word order (*apokoinou*). The vertical bars show the leftmost and rightmost limits of grammaticality of the two sentence patterns.

(3) a. That's | the only thing he does | is fight.

b. Willemse heeft | gisteren | heeft de dokter nog gezegd dat het mag.
(Willemse has yesterday has the doctor said it is allowed.)

c. Ik ben | binnen vijf minuten | zijn we terug.
(I am within five minutes we are back)

While conceptual deletion and replacement seem to be relatively infrequent as causes of incrementation or correction, *addition* is frequent. It is assumed that conceptual processing, just like syntactic processing, takes place in a piecemeal way in spontaneous speech, so that the continual addition of conceptual fragments to existing ones is quite normal. Addition can be of two kinds. The first kind is an addition of a conceptual fragment which is to be in *conjunction* or *disjunction* with an existing fragment and thus leads to a syntactic coordination, as in (4a,b).

(4) a. Last night I saw John ... and Mary.

b. It must have been John ... or Mary.

The second kind is the addition of a new conceptual fragment in any other relationship than conjunction or disjunction. This may give rise to an *expansion*, i.e., the current utterance is continued with a syntactic fragment which is not a member of a coordination but has some syntactic relation (such as subject, direct object, modifier, etc.) to the current utterance or part of it. Simple examples are (5a,b).

(5) a. John and Mary ... went to a party.

b. John and Mary went ... to a party.

It is almost self-evident that the order of successive conceptual inputs is an important factor in determining left-to-right order of constituents. This has indeed been experimentally confirmed by Schriefers and Pechmann (1988; see also Section 2.1). E.g., the difference between (6a) and (6b) can in certain contexts be attributed to the fact that the concept *white* is accessible to the Formulator earlier, resp. later than *big*.

(6) a. A white big triangle.

b. A big white triangle.

After a conceptual addition, it may not always be syntactically possible to continue a partially uttered sentence. Lexico-syntactic restrictions may severely limit the possible ways of expanding the syntactic structure or finding an appropriate word order. In English, for example, it seems impossible to expand (7a) to express a conceptual increment corresponding to *likes to*, as in (7b). By contrast, an equivalent expansion is possible in Dutch, where the meaning underlying *likes to* can be expressed by means of an adverbial phrase as in (7c).

(7) a. John comes ...

b. John likes to come.

c. Jan komt ... graag.

The difference between the English and the Dutch example shows that the restrictions are language-specific and lexico-syntactic in nature. In circumstances where expansion is impossible, the monitor will receive no output from the Grammatical Encoder. A syntactic dead end will thus be detected and a self-correction will be triggered by causing the conceptual structure to re-enter the Grammatical Encoder to be reformulated:

(8) John comes ...uh... likes to come to the party.

Another example of an impossible expansion in English is the expansion of (9a) to (9b). However, the apposition in (9c) or the relative clause in (9d) offer alternatives. There may be a covert self-correction during the grammatical encoding of these sentences, which in the utterance is possibly marked by a pause.

(9) a. The man ...

b. The bald man ...

c. The man ... the bald one that is, ...

d. The man ... who is bald, ...

Syntactic dead end is of course but one possible cause of self-correction. Other types of errors which are detected by the monitor and may result in a self-correction include the wrong choice of lexical material, fusion errors, and articulation errors. It is often unclear whether in a particular utterance, e.g. (2a), the cause of the correction is a conceptual modification or the detection of a lexical error. A discussion of these phenomena is beyond the scope of this chapter. The question of how much conceptual material re-enters the Grammatical Encoder to produce a self-correction is an interesting one, but it will likewise not be discussed here (see Van Wijk & Kempen, 1987, for some relevant findings and ideas). My present aim is to show the global picture of the relations between incremental conceptualization and self-correction.

Figure 1.3 gives a schematic overview of the conceptual and monitoring processes discussed in this section. The process flow is downward. Non-retracing repairs and normal incrementation are grouped together in this overview.

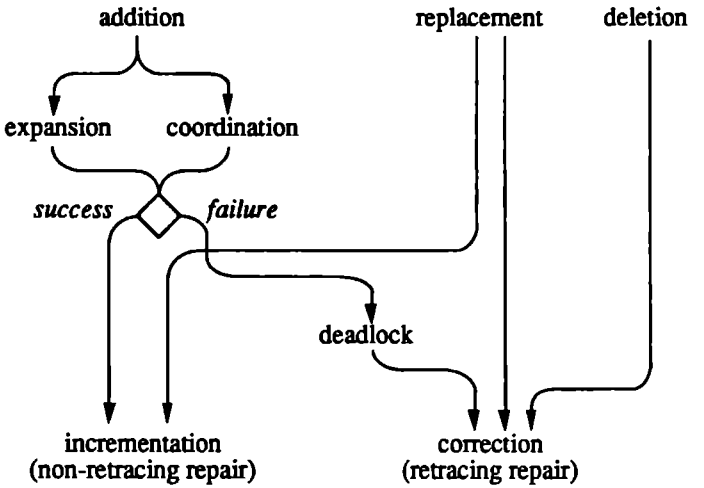


Figure 1.3
Conceptual modifications, monitoring, and consequences for grammatical encoding

1.2 Mechanisms for incremental grammatical encoding

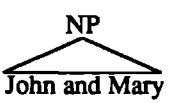
In this section, the three types of mechanisms in grammatical encoding (expansion, coordination and correction) are discussed in more detail.

1.2.1 Expansion

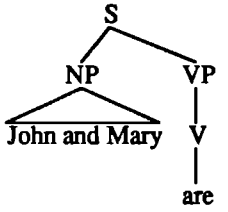
Three kinds of expansion are distinguished, depending on the location in the syntactic structure where a new syntactic fragment is added. *Upward* expansion causes the syntactic tree to grow upward, i.e., the original root node is no longer the root node of the expanded tree. Other cases are called *downward* expansion, when new branches are added below an existing node. Finally, there is a special case called *insertion*, if syntactic material is inserted between existing nodes. The sequence (10b-e) shows roughly how the various kinds of expansions affect a syntactic tree for utterance (10a).

(10) a. John and Mary are at the party ... seem to be at the party.

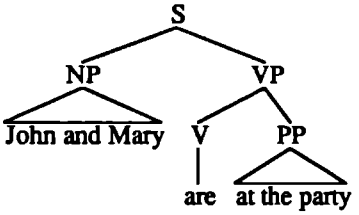
b.



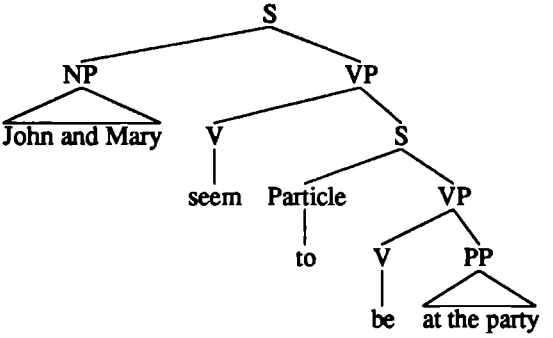
c.



d.



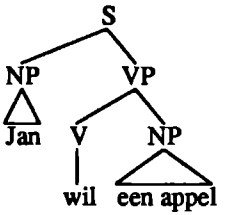
e.



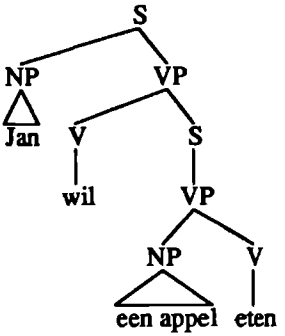
Insertion does not necessarily lead to an overt self-correction, as was the case in (10). An example where insertion leads to the continuation of a fragment which has already been uttered is the Dutch sentence (11a). The English translation contains a correction, but the Dutch original does not. The insertion is depicted in the sequence (11b,c).

(11) a. Jan wil een appel ... eten.
(John wants an apple ... wants to eat an apple.)

b.



c.



If upward expansion is allowed, then one must also allow situations where an initial conceptual fragment does not lead to the construction of a main clause. Instead, an isolated noun phrase may be produced, as in (12).

(12) He ...

Such an initial constituent is 'unattached' in the sense that it does not have a syntactic relation to a mother node. Although a subsequent conceptual fragment may cause the construction of a mother node, it would be a handicap if uttering the initial constituent had to be postponed until a syntactic relation was assigned to the constituent. However, how should the Grammatical Encoder make decisions which depend on such a syntactic relation, for instance, choosing the surface case marking (*he*, *him*, *his*), while that relation has not yet been specified? One possible solution consists in carrying out one or more *provisional* upward expansions until a sentence node has been created. Subsequent conceptual fragments may lead to syntactic fragments which are *actual* upward expansions. The system then attempts to combine the actual syntactic nodes with the provisional ones. If this combination, which can be achieved by means of *unification* (Kay, 1979), is successful, the nodes are merged. This leads to a successful expansion, as in (13a). Unification will fail when the syntactic functions in the provisional and the actual expansions of nodes are different. In that case, either a *restart* using a different syntactic structure may take place (13b), or lexico-syntactic alternatives may be explored, e.g. passivization (13c).

(13) a. He ... left.

b. He ... They invited him.

c. He ... was invited.

What heuristics or preferences does the Grammatical Encoder use when choosing between alternative possibilities for provisional upward expansions? A partial answer is provided by Bock and Warren (1985). They establish a relationship between

conceptual accessibility (the ease of retrieving conceptual information from memory) and the hierarchy of grammatical relations which plays a role in various cross-linguistic and within-language phenomena (Keenan & Comrie, 1977):

(14) subject > direct obj. > indirect obj. > oblique > genitive > obj. of comparison

Similar results were obtained in a sentence recall experiment performed by Keenan & Hawkins (1987). My hypothesis is, that this (or a similar) hierarchy plays a role as a preference scale in the incremental generation of sentences. The first constituent which is to have a syntactic relation in a sentence will have a higher probability of being realized as a subject than as direct object, etc., according to the hierarchy. Subsequent fragments may find the relations higher in the hierarchy already occupied by previously created constituents and will be assigned a function lower in the hierarchy. Since the hierarchy is correlated with preferred word order, it thus serves to guide the sentence formulation process toward maximally fluent incremental sentence generation.

Other factors may complement the use of the relational hierarchy in incremental sentence generation. One possible factor consists of the *conceptual* category of the initial fragment. E.g., the preferred function assigned to a time-indicating NP such as (15) may not be *subject* but some lower member of the hierarchy such as *sentence modifier* (oblique).

(15) Monday morning, ...

1.2.2 Coordination

Coordination is viewed as an iteration of the grammatical encoding process on several conceptual fragments which are linked to each other as members of a conjunction or disjunction. The result of lexicalizing and formulating these is a special phrase called a *coordination* which has a number of conjuncts as its immediate constituents.

Often coordinations come about in a piecemeal way because the speaker may keep adding conjuncts, even after some have been uttered. Conjuncts are often realized with 'comma intonation' as long as there is at least one further element to be formulated. If it is the final element, it is added after insertion of a conjunction word like *and*. But utterances in which 'afterthoughts' give rise to multiple occurrences of the conjunction word (16a) or even *right dislocations* (16b) are not unusual in spoken language. These are accounted for by assuming that new descriptions have entered the Formulator when it had already finished a conjunction.

(16) a. John, Peter and Mary ... and Anne came home.

b. John, Peter and Mary came home ... and Anne.

1.2.3 Self-correction

Self-corrections are governed by rules which determine how much of the original utterance needs to be repeated. For example, (17a) is not well-formed because some obligatory constituents are missing: all constituents in the self-correction which are to

the right of the replaced main verb should be reformulated, as in (17b). Likewise, (18a) is not grammatical because the entire NP should be reformulated (18b).

(17) a. *You should have sent that letter ...uh... handed over.

b. You should have sent that letter ...uh... handed it over.

(18) a. *Tony is baking a cake ... sugar-free.

b. Tony is baking a cake ... a sugar-free cake.

Levelt (1983:78) has observed that the rule which speakers obey when deciding how far they should retrace is similar to the retracing rule for coordinations. He then stated a *well-formedness rule* for repairs in terms of the grammaticality of coordinations, linking the ill/well-formedness of (17a,b) to that of (19a,b) respectively.

(19) a. *You should have sent that letter or handed over.

b. You should have sent that letter or handed it over.

Following Levelt's rule, a mechanism for generating self-corrections could have the same underlying principles as the mechanism for coordination. If an error has been detected by means of monitoring and its cause has been diagnosed (dead end, conceptual replacement, lexicalization error, etc.), a conceptual fragment marked as the correction of some earlier fragment is entered into the Grammatical Encoder. The correction marker is treated by the Grammatical Encoder in much the same way as a conjunction marker, the only difference being that it is realized as a pause or as a correction term (such as *uh*), rather than as comma intonation or a conjunction (*and, or*).

Example (20a) shows that self-correction and coordination can occur in one and the same constituent. In addition, examples (20a,b) illustrate that the ambiguity of certain self-corrections is similar to that of corresponding coordinations, which again suggests that they should be treated in a similar way.

(20) a. Peter and Mary ...uh... John left the house.

b. Peter and Mary or John left the house.

However, Van Wijk and Kempen (1987), who have verified Levelt's well-formedness rule, found that it covers only one type of self-corrections, which they call *reformulation*. Self-corrections of another type, which they call *lemma substitution*, e.g. (21a), do not require the computation of a new syntactic structure, because simply replacing a lemma in the existing structure suffices. Other self-corrections are really *restarts*, i.e., instead of carrying out a repair, a whole utterance is rejected and the speaker starts all over, e.g. (21b).

(21) a. Do you really want to buy that record ...uh... compact disc?

b. Did the student ...uh... Did you ask the student anything?

The choice between correction strategies made by the Grammatical Encoder seems to be partially dependent on the origin of the correction. Van Wijk and Kempen found that conceptual addition often leads to reformulation while replacement and deletion often trigger lemma substitution. In addition, it seems that restarts are often caused by syntactic dead ends.

1.2.4 Control structure

Because deleting, replacing and adding conceptual material may occur repeatedly and independently of each other, the various mechanisms of the Grammatical Encoder, namely self-correction, coordination and expansion, may occur in one utterance and may even be embedded in one another. For example, a conceptual addition may cause a coordination; within one of the conjuncts, a conceptual addition may lead to an attempt at expansion, which, if unsuccessful, will cause a correction to occur, etc. An annotated example of such a sequence is (22).

(22) Peter ...
and that woman ... (*conjunction*)
who sleeps ... (*downward expansion*)
who never sleeps more than five hours a night ... (*downward expansion with retracing*)
or even less ... (*disjunction*)
came early to my party. (*upward expansion*)

A sequential, single-process Grammatical Encoder might be based on a control structure with nested iteration loops on the output of the Conceptualizer. One loop is expansion, which may cause the addition of mother or daughter nodes in the syntactic tree. The other loop combines correction and coordination. It may iterate within each constituent, where it causes the addition of coordinated elements or corrections. Each of the two loops may be nested within the other one.

However, grammatical encoding need not be realized in a sequential process. The next section will introduce an architecture for sentence generation based on a parallel control structure.

1.3 IPF: an Incremental Parallel Formulator

This study is mainly concerned with grammatical encoding, which is the machinery deployed in constructing syntactic structures expressing a speaker's intention. IPF (Incremental Parallel Formulator) is a computational model of a psychologically plausible Formulator. IPF does not currently account for all mechanisms discussed in Section 1.2 but handles expansions in upward as well as downward directions. The following section outlines the parallel architecture of IPF.

1.3.1 Parallelism in incremental generation

Above it was suggested that the main modules involved in language generation, the Conceptualizer, the Formulator and the Articulator, operate independently of each other and in a piecemeal fashion. These modules are relatively autonomous components in the generation system. They meet the requirement of *informational encapsulation* in the sense that their input is of a maximally restricted sort and their mode of operation minimally affected by other components (Fodor, 1983; Levelt, 1989). The Formulator is only provided with preverbal messages (conceptual structures) and the Articulator only with phonetic representations. There is neither direct feedback from the Articulator to the Formulator, nor from the Formulator to the Conceptualizer.

It was further suggested that these autonomous modules operate *in parallel* on different parts of the utterance. Since it is likely that formulation and articulation are highly automatic nonintentional processes (at least in skilled speakers of a language), no division of attention is needed to perform these tasks in parallel with other tasks.

The communication between the modules operating in an incremental fashion can be modeled in different ways. Hoenkamp (1983:114-117; also De Smedt & Kempen, 1987) models the communication channels between the modules as *streams*: conceptual fragments are entered at one end of a stream and are 'consumed' by the Formulator at the other end, as shown in Figure 1.4. While the Formulator is processing elements from the stream, the Conceptualizer runs simultaneously, planning more content and thus adding more elements to the end of the stream. The conceptual fragments contain markers which indicate their relationship to fragments earlier in the stream.

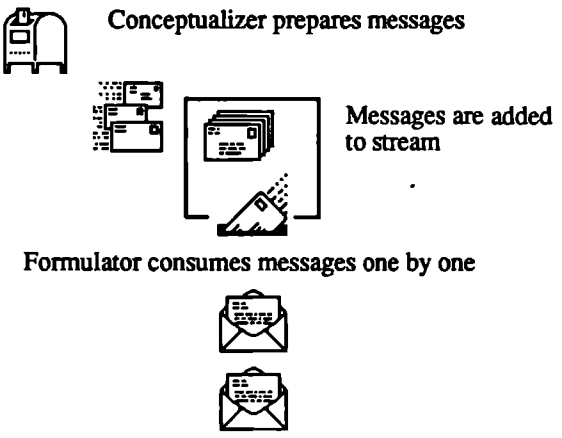


Figure 1.4
Stream of conceptual fragments

However, the Formulator need not operate in a sequential fashion, but may be viewed as a system which is *distributed* over several parallel processes. Several

conceptual fragments which have entered the Formulator may be processed simultaneously: each conceptual fragment immediately spawns an independent Formulator process, as shown in Figure 1.5. Some speech errors of the exchange type (Garrett, 1975) reflect this form of computational simultaneity within the grammatical encoding stage.

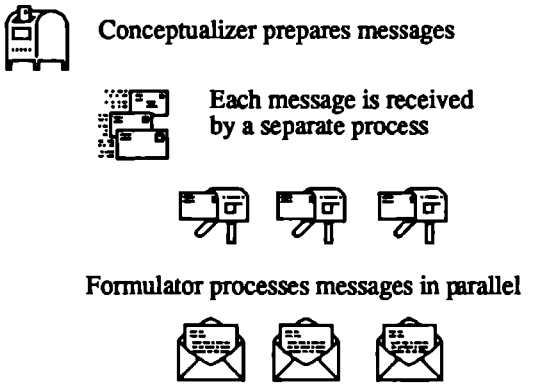


Figure 1.5
Parallel processing of conceptual fragments

The distribution of the formulation task among a number of parallel processes, each of which is responsible for the construction of part of the syntactic structure, may significantly improve the performance of the Formulator. The possibility of parallelism in the Formulator has already been recognized by Kempen and Hoenkamp (1987; also Hoenkamp, 1983). However, there are several questions which they do not address. The following section will raise some of those questions and indicate how this study claims to answer them.

1.3.2 Incremental sentence generation: some research questions

The first part of this study deals with psychological and linguistic aspects of incremental sentence generation. It may be a good starting point to survey briefly the research in the field (Chapter 2). For an incremental generator it seems inevitable to determine first what the units are which are passed from one stage to the next. In fact, psycholinguists have long been concerned with the following question:

What are the units of grammatical encoding?

Although experimental research has not yielded definitive answers to this question, Pechmann (1989) finds it very likely that people can add syntactic increments corresponding to single phrases and even single words. IPF therefore attempts to model grammatical encoding in a relatively fine-grained way. Is it necessary for this goal to build a new computer model from scratch if so many natural language processing systems have been built? The question can in general be stated as follows:

What is wrong with current natural language generation systems?

It appears that few natural language generation systems are meant to simulate the human speaker under realistic psychological constraints. In particular, the incremental mode of generation has been underexploited. Moreover, current linguistic formalisms are not oriented toward the manipulation of partial sentences. It is therefore necessary to work on new models of language generation as well as new grammar formalisms which support incremental generation. In addition, the use of an incremental sentence generation system need not be restricted to psycholinguistic modelling. If one considers the question:

What are the applications of incremental sentence generation?

one finds that—surprisingly perhaps, an incremental and parallel mode of generation is not only useful but almost obligatory for certain practical applications. Examples are real-time speech production, simultaneous interpretation, multi-modal user interfaces, and real-time interpretation of visual scenes.

Given that speakers can compose an utterance in a piecemeal fashion, I will now turn my attention to some requirements of an incremental mode of generation on the grammar formalism (Chapter 3). A first question concerns upward and downward expansion:

How can a syntactic structure be constructed from the bottom of the syntactic structure upward as well as from the top downward?

A top-down formalism, such as a rewriting system based on PS rules, or a hierarchy of procedures (Hoenkamp, 1983) is unpractical. A solution lies in the adoption of a *unification*-based formalism. Since unification operates irrespective of the orientation of syntactic segments, a structure can be composed from its substructures in any order. Having dealt with expansion in the vertical direction, I must also deal with structural growth in the horizontal direction, since many formalisms incorporate all constituents of a nodes at the same time:

How can sister nodes be incrementally added to an existing phrase in the syntactic structure?

An solution may consist of the specification of individual Immediate Dominance (ID) relations between nodes, by factoring out lexico-syntactic restrictions on sisterhood. A third requirement for a grammar formalism concerns word order. Given that the order of conceptual inputs does not necessarily correspond to the eventual left-to-right order of corresponding nodes in the resulting utterance, it seems unpractical to work within a formalism which directly attempts to construct an ordered structure. Therefore the question arises:

How can a grammar incrementally incorporate new nodes in the structure while making separate commitments with respect to their left-to-right ordering?

An answer may consist of factoring out knowledge about Linear Precedence (LP) from ID relations. Thus the basis of a grammar formalism for incremental generation could be formed by the specification of single ID relations with a separate specification of their possible sisters and LP constraints.

Kempen (1987) proposes a new formalism which meets these requirements. Syntactic structures are constructed out of so-called *syntactic segments* which each represent a single ID relation between nodes. A segment consists of two nodes representing grammatical categories, and an arc representing a grammatical function. They are graphically represented in vertical orientation, where the top node is called the *root* and the bottom node the *foot* (see (23c) for an example). Segments join to form a syntactic structure by means of a general *unification* operation.

This formalism, which Kempen originally called *Incremental Grammar*, is especially suited to—but not restricted to—incremental generation. Since individual segments can be added to an existing structure, the formalism is suited for incremental generation at a fine-grained level, i.e., by individual constituents. In order to distinguish clearly between the grammar formalism and the processing model, the grammar formalism will be renamed *Segment Grammar* (*SG*). This thesis works out SG in more detail (Chapter 3). I want to address the following questions:

> *How does SG employ the unification operation and how does it differ from other unification-based formalisms?*

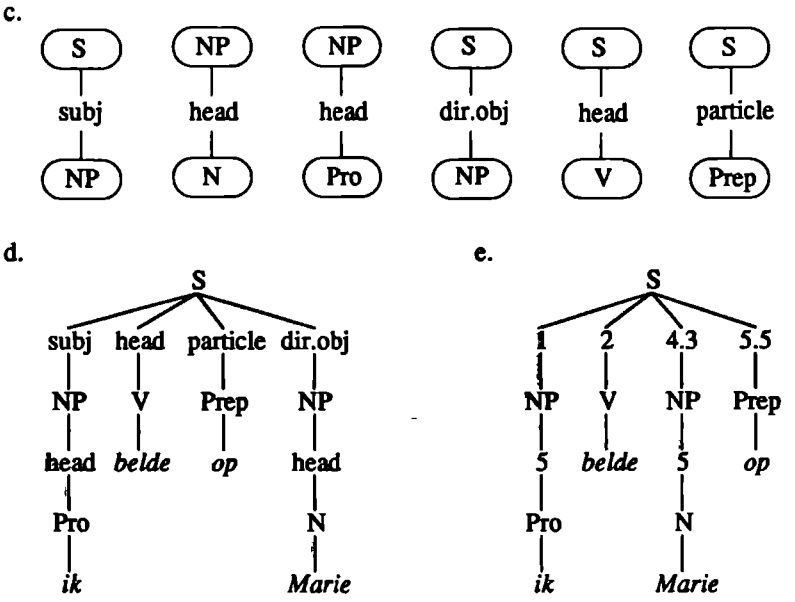
In response to this question, it will be shown how unification in SG is a local operation on nodes which does not involve the choice (alternation) mechanisms of other unification-based formalisms. In addition, the following question is addressed:

> *Which syntactic structures are involved in linguistic description using SG and how are they incrementally constructed?*

I believe that the various syntactic relations which come into play at the grammatical encoding stage can best be represented by the construction of two subsequent structures—*functional structures* (or *f-structures*) and *constituent structures* (or *c-structures*). F-structures represent grammatical relations between syntactic elements while c-structures represent 'surface' constituency and word order. By way of example, the f-structure (23d) for Dutch examples (23a) as well as (23b) consists of six segments (23c). The assignment of left-to-right positions to constituents is modeled as the derivation of a different kind of structure—a *c-structure*. By way of example, c-structure (23e) is assigned to (23a). Left-to-right order of constituents is indicated by means of a number system.

(23) a. Ik belde Marie op. (I called up Mary)

b. Marie belde ik op. (Mary I called up)



Having decided on SG as a formalism for incremental generation, I will then turn my attention to processing aspects (Chapter 4). The following architecture is proposed. The construction of f-structures is driven by conceptual messages which are input to the Formulator. F-structures give rise to c-structures, which are input to the Phonological Encoder; the resulting phonetic strings are eventually uttered as speech sounds. A schematic overview of the formulation process is given in Figure 1.6.

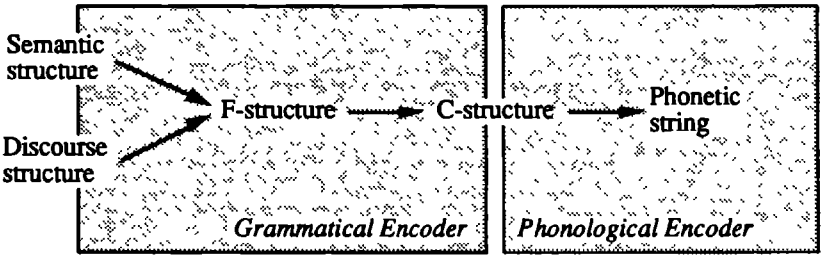


Figure 1.6
The generation of subsequent linguistic descriptions during the formulation stage

Little attention will be given to the origin of the semantic structures which are input[2] to this machinery. But to demarcate the scope of the computer model, I must first address the question:

[2] Discourse structures are also input to the Formulator. However, since IPF at present handles only individual sentences, discourse information is not taken into account in the present model, except in the disguise of features such as *definiteness*.

If it can be assumed that planning units can correspond to a single phrase and even a single word, what are the conceptual messages that are input to the Formulator?

It will be assumed that the Conceptualizer packages conceptual information into three types of messages which enter the Formulator:

1 *Semantic concepts*. These are references to entities, events, etc. in the domain of discourse which are to be referred to in the utterance.
2 *Case relations*. These are 'deep' cases expressing semantic roles between concepts. There is no special meaning attached to the case labels; they simply serve to distinguish which participants in the situation are expressed.
3 *Features*. For simplicity, it is assumed that these are prepared in a rather language-specific form and can thus readily be used as syntactic features. Examples are definiteness, number, etc.

On the output side of the Grammatical Encoder, the details concerning the realization of surface structures as phonologically specified strings will not be dealt with here. The output of the Grammatical Encoder consists of c-structures which are taken as input by the Phonological Encoder, presumably in a left-to-right fashion. C-structures contain ordered and syntactically specified words and are incrementally derived from f-structures. During generation they are provided with a constellation of features which allow the Phonological Encoder to produce ordered and phonologically specified strings.

The relatively small size of each conceptual input, together with the adoption of SG, lead to considerable flexibility in the generator. This flexibility can be exploited via the time dimension in a parallel system, if each process starts, runs and finishes independently of the others:

How does the passing of time in a distributed Grammatical Encoder affect the shape of an utterance?

If we assume, with Hoenkamp (1983:18), that incremental generation is guided by the principle "What *can* be uttered *must* be uttered immediately", then the order in which the Formulator finishes the construction of syntactic constituents will exert an influence on the order in which they are uttered. Surely this is not the only determining factor: language-specific word order restrictions will in general prevent constituents from being uttered when they do not form a grammatically well-formed sequence. Thinking backwards, the Formulator can only start working on a conceptual fragment when that fragment is made accessible by the Conceptualizer. Extending the principle of incremental generation so that "What *can* be formulated *must* be formulated immediately", we deduce that the order in which conceptual fragments are passed on to the Formulator is—indirectly—also a factor affecting the shape of an utterance.

It is important that concepts and case relations can be entered into the Formulator *individually* rather than grouped in a large conceptual structure. This allows their manipulation in terms of ordering and spacing in the time dimension. Time delays between conceptual inputs may simulate the different conceptual accessibilities of these

inputs. The more time there is between successive inputs, the more sequential their processing will be and the more their initial ordering will be reflected in the eventual utterance. Entering successive inputs shortly after one another will cause more overlap in their processing, as if a whole chunk of conceptual material has entered the Formulator.

Summing up, a claim of this thesis is that the processes which together make up the grammatical encoding stage can be viewed as *competing*, i.e., in a *race* with each other. The finishing of each process is affected mainly by (i) the time of input and (ii) the computational load of the process. The outcome of the race shows up at the surface level in the form of lexical choices and word order variations.

Although it would be too far-fetched to discuss the impact of an incremental mode of processing on all possible syntactic constructions, I nevertheless want to treat a special class of constructions in Dutch. In particular,

> *If sentence formulation proceeds by the addition of syntactic constituents to a structure, how are* discontinuous constituents *explained?*

The separation of functional (grammatical) relations between syntactic constituents on the one hand, and left-to-right order of constituents on the other hand, can account for several types of discontinuous constituency. A treatment of discontinuities in the SG framework is presented in Chapter 5. A similar separation between f-structures and c-structures has been argued for by proponents of Lexical-Functional Grammar (LFG; Kaplan & Bresnan, 1982), but they do not commit themselves to a theory of linguistic *processing* with such a framework. IPF, in contrast, proposes a specific mechanism to build syntactic structures.

Natural language generation is a knowledge intensive task. Most of Part Two will therefore be concerned with the questions, how knowledge about language can best be represented to build an efficient and flexible system, and what processes operate on this knowledge. The linguistic knowledge in SG is organized in a large number of linguistic structural units—categories and segments. These form a lexicalized grammar—a grammar where syntactic constraints are connected to lexical categories. Given the tremendous size of the lexicon, there is another interesting research question:

> *How can massive redundancy be avoided in the lexicon (in particular, in the lexicon associated with a Segment Grammar)?*

This question is answered by the adoption of *default inheritance* as an important mechanism in linguistic knowledge representation. Using inheritance, specialization hierarchies of linguistic concepts such as syntactic segments, phrases and words can be built. To this end, object-oriented programming techniques are proposed (Chapters 6 and 7). By viewing an object as a specialization of another, the shared knowledge need only be specified in the more general object. Also, new linguistic objects can be composed out of the combined knowledge in several other objects. Object-oriented representation is joined to unification by representing feature structures as objects. Chapter 8 presents an implementation of SG in CommonORBIT, an object-oriented language.

Finally, I address some computational questions associated with a parallel architecture (Chapter 9). The potential of SG for parallel processing is tapped by allowing unifications to happen in parallel, and the following question cannot be avoided:

> *How can syntactic coherence be guaranteed if grammatical encoding is distributed among several independent parallel processes?*

A solution consists of embedding the unification in appropriate control structures so that a single node cannot be involved in more than one unification at a time. In this way, parallelism is restricted to operations on distinct parts of the syntactic structure. Concurrent programming techniques will be used for this purpose.

1.4 Concluding remarks

We have seen how incremental generation and self-corrections can be accounted for by allowing increments and other conceptual modifications after the grammatical encoding process has already started. It is assumed that the different modules which are involved in sentence generation (i.e., Conceptualizer, Formulator and Articulator) can run in parallel. Three types of conceptual modifications may occur while formulation is already under way: deletion, replacement and addition. Deletion and replacement of a conceptual fragment which is already being formulated, typically give rise to a self-correction. Addition may give rise to a coordination or an expansion. Of the latter there are several types: upward and downward expansion (and possibly a mixed case called insertion). A monitor inspects the results of the generation process, which allows the detection of errors. One such error, dead end, occurs when it is impossible to continue a syntactic fragment with the desired increment. Upon the detection of errors, self-corrections may be triggered.

Although it is fairly obvious that human speakers generate sentences incrementally, many empirical questions remain. It is not clear how large the conceptual units are that can be generated independently and in a piecemeal fashion. While some researchers claim that these units are fairly large, others have found that they can also be rather small. The simulation model which will be presented in the present study supports a relatively fine-grained incremental strategy: the Grammatical Encoder allows the addition of single branches and nodes in a syntactic tree. Few current natural language generation systems allow incremental generation at such level of detail. In addition, it is assumed that the Grammatical Encoder is distributed among a number of parallel processes, each of which operates independently on part of the syntactic structure.

A natural language generation system which operates according to these assumptions must be supported not only by an incremental and parallel generation strategy, but also by a grammar formalism which is designed to meet the constraints imposed by its incremental and parallel architecture. This will be an important consideration in Part One.

Part One: Psychological and linguistic aspects

The first part of this study discusses psychological and linguistic aspects of incremental sentence generation. A computational model of grammatical encoding is described which allows upward as well as downward expansions of the syntactic structure. It is shown how the generation task can be simulated under the constraints of an incremental mode and grammatical encoding can be distributed among parallel processes. In addition, it is shown how syntactic structures built by the model can account for certain linguistic phenomena.

Chapter 2 is a review of related research on sentence generation. It briefly reviews psycholinguistic research on the question what the units of grammatical encoding are. Then an overview is given of some natural language processing systems which generate in a partly incremental fashion. The chapter concludes with a review of some application-oriented work.

Chapter 3 describes and further develops *Segment Grammar* (SG), a unification-based grammar formalism which is promoted by Kempen (1987) as especially suited to incremental sentence generation. Syntactic segments relate two nodes (a root and a foot) by means of a grammatical relation. Sentence generation is seen as the unification of nodes of several segments in such a way that a coherent structure is formed. This chapter provides a formal definition of SG as well as an informal synopsis of its principles.

In Chapter 4, the Grammatical Encoder is presented in detail. It is shown how *f-structures* in SG are generated in a lexically guided manner, with unification as a general structure building operation. A lexicalization process creates content words while a functorization process adds function words. The lexicon is a phrasal lexicon augmented with lexical rules which may derive new lexical entries from existing ones. While the f-structure is constructed, an ordered constituent structure (a *c-structure*) is incrementally derived from it. Word order is assigned to constituents independently of one another by a system of absolute left-to-right positions.

Chapter 5 treats discontinuous constituency from the perspective of incremental generation using SG. The assignment of left-to-right positions may be attended by changes in the immediate dominance relations. Thus, a c-structure need not be isomorphic with the corresponding f-structure for a sentence. It is shown how this mechanism can account for various kinds of discontinuous constituents including right dislocation, S-O raising, and unbounded dependencies. The well-known cross-serial dependencies in Dutch are also accounted for.

2 Probing an incremental mode of language generation

By means of introspection, speakers may become aware of the fact that they often produce speech incrementally. Von Kleist (1805) was possibly the first to write about his awareness of incremental sentence generation:

> "Aber weil ich doch irgend eine dunkle Vorstellung habe, die mit dem, was ich suche, von fern her in einiger Verbindung steht, so prägt, wenn ich nur dreist damit den Anfang mache, das Gemüt, während die Rede fortschreitet, in der Notwendigkeit, dem Anfang nun auch ein Ende zu finden, jene verworrene Vorstellung zur völligen Deutlichkeit aus, dergestalt, dass die Erkenntnis, zu meinem Erstaunen, mit der Periode fertig ist. Ich mische unartikulierte Töne ein, ziehe die Verbindungswörter in die Länge, gebrauche auch wohl eine Apposition, wo sie nicht nötig wäre, und bediene mich anderer, die Rede ausdehnender, Kunstgriffe, zur Fabrikation meiner Idee auf der Werkstätte der Vernunft, die gehörige Zeit zu gewinnen. Ich glaube, dass mancher grosse Redner, in dem Augenblick, da er den Mund aufmachte, noch nicht wusste, was er sagen würde. Aber die Überzeugung, dass er die ihm nötige Gedankenfülle schon aus den Umständen, und der daraus resultierenden Erregung seines Gemüts schöpfen würde, machte ihn dreist genug, den Anfang, auf gutes Glück hin, zu setzen." (Von Kleist, 1805)[1]

In the foregoing quote, Von Kleist suggests that speakers may start uttering a sentence while they barely know what they are going to say. Having uttered the first words, speakers use various means to stretch time in order to complete the conceptual content to be incorporated in the sentence. Incremental generation is necessitated by the pressure on speakers to produce speech in time. Indeed, it is a well known fact that long pauses in monologues are considered disruptive, and that pauses in dialogues are opportunities for turn taking. Incremental generation also reduces the memory load, because it is not necessary to hold a whole sentence in memory before starting the utterance. Assuming true parallelism, incremental generation may substantially speed up the generation process.

However, even if incremental generation itself is an undisputed fact from experience, it is impossible to find by introspection on what level of detail the incremental architecture proposed here corresponds to human sentence processing. This chapter will review some experimental work which was carried out in search of the units of

[1] This text was brought to my attention by Thomas Pechmann.

grammatical encoding. After that, some related work in Computational Linguistics will be discussed. Finally, I will discuss some applications of incremental sentence generation.

2.1 Units of grammatical encoding

Although there has been some experimental work related to the questions what the levels and units of sentence generation are in spontaneous, unprepared speech, there are few definitive conclusions to be drawn from the results. Perhaps there are several possible planning units (Levelt, 1989:23-24). Still, if we want to build a computational model of language generation, we must find where to draw the line.

Early psycholinguistic research investigated pauses between clauses in spontaneous speech in an attempt to identify the fundamental units of syntactic planning. The fact that many surface clauses are preceded by pauses seemed to support the hypotheses that there is only one level of syntactic processing and that the clause is the basic planning unit. Goldman-Eisler (1972) postulated that sentences, and to some extent, clauses, are highly cohesive units which reflect thought units. Postulating such large units is opposed to the idea that the sentence is the linguistic expression of the fact that a connection of *several* more or less independent ideas has occurred in the mind of the speaker (Paul, 1909).

However, these hypotheses were refuted by other research. Brotherton (1979) found many fluent runs of three or more clauses without any pausing in a large corpus of spontaneous speech, thus showing that pauses are not necessary for planning speech. Also, Boomer (1965) had already found that most hesitations occur after the first word of a phonemic clause. He explained this by postulating that the selection of the first word "has in greater or lesser degree committed the speaker to a particular construction or at least a set of alternative constructions, and has also foreclosed the possibility of other constructions". Boomer thus implies that there is incremental syntactic planning *within* the limits of the clause.

Later research tried to measure cognitive activity *during* speech. Ford and Holmes (1978) conducted a dual-task experiment to investigate the hypothesis that speakers articulate one clause while planning the next one. Their subjects were given the task of monitoring for tones while they were being interviewed. The reaction times (RTs) thus obtained are supposed to vary inversely with 'spare' processing power at the time of the stimulus presentation and thus measure the cognitive burden of speech planning at that point. Ford and Holmes found that in multi-clause sentences, RTs were indeed significantly longer toward the end of the first clause, which supports their hypothesis.

Petrie (1989) suggests that speakers may prepare the next clause while simultaneously uttering the current one, but if the clause is too complex, they will pause between clauses to finish the required planning. In fact, Boomer already gave a similar explanation of pauses between clauses:

> "As a given clause is being uttered the next one is taking shape and focus. At the terminal juncture the next clause may be ready, in which case it will be uttered fluently, as were more

than half the phonemic clauses in this corpus. If, however, the emerging clause has not yet been subjectively formulated, speech is suspended until the entire pattern is clarified." (Boomer, 1965)

From this hypothesis, Petrie predicts that:

"(a) RTs will be significantly higher towards the ends of clauses which are followed by pauses before the next clause than towards the ends of clauses which have no pause before articulating the following clause; and (b) clauses which have pauses before them will be syntactically more complex than those which have no pause." (Petrie, 1989)

The hypothesis that the next clause is planned while the current one is being articulated also accounts for the fact that the interacting units in speech errors often belong to two adjacent clauses: the elements of two units which are simultaneously processed can easily be confused. Beyond adjacent clauses, however, there should be no interaction.

In order to investigate these assumptions, Petrie has further developed the dual-task experimental technique used by Ford and Holmes (1978). The dual-task of monitoring for tones while speaking, which was used by Ford and Holmes, measures only overall cognitive processing. However, Petrie wanted to measure specific kinds of processing, in line with the more recent hypothesis that the human mind consists of a number of specialized processors (Allport, Antonis & Reynolds, 1982). She developed a dual-task which required semantic or syntactic processing while speaking. Subjects were made to monitor for pairs of words on a video display while they were speaking. In one version of the experiment, they had to decide whether the target is a member of the given semantic category (e.g. ANIMAL RABBIT). In another version, they had to decide whether two words can follow each other in a grammatical English phrase (e.g. TO HAPPY). The data gathered so far seem to support her hypothesis, although they are still not fully conclusive (Petrie, personal communication).

Petrie's experiments have two limitations. First, she only investigates the clause as a possible planning unit, following Ford and Holmes' example. Second, the dual-task does not shed light on the data flow between processing levels of the generation process, because there is no control over the input to each level.

These limitations can be overcome by giving the experimenter control over the conceptual input to the formulation process. Schriefers and Pechmann (1988; also Pechmann 1987; 1989) therefore developed an experimental paradigm based on a *referential communication task*. Subjects were asked to give a verbal description of a given target object to distinguish it from a number of other, simultaneously displayed objects. The descriptions collected in these experiments were in the form of NPs possibly including a shape category and the features color and size, e.g. WHITE BIG TRIANGLE.

Inspection of the resulting descriptions revealed irregularities which can be explained by non-optimal planning, in particular that speakers indeed start formulating before they have fully determined some set of distinguishing visual features. In addition to overspecified—as opposed to minimally specified—referential descriptions, Pechmann (1989) also reports non-standard order of the features color and size as

prenominal adjectives—an indication that speakers favor an order depending on the piecemeal conceptual availability of the features.

To conclude, Pechmann finds strong evidence that people produce incrementally in a fine-grained way, i.e., not only in clause units, but also in units corresponding to single phrases and even words. This refutes Ford and Holmes' (1978) suggestion that "it seems likely that the [planning] unit is larger than the single word". Pechmann's experiments show that incrementation by single words does occur, although effects due to phrase-sized chunks are also found.

2.2 Natural language processing systems

Most natural language generation systems in the fields of AI and Computational Linguistics have not been designed for the simulation of spontaneous speech but for the construction of carefully planned sentences and texts. Hence, it will not be surprising that in most systems the conceptual and lexico-syntactic stages are ordered strictly serially for a complete sentence. However, some attention has been given to incremental generation in at least two other systems: MUMBLE and KAMP.

In MUMBLE (McDonald & Pustejovsky, 1985a), a *conceptual planner* (Conceptualizer) and a *linguistic module* (Formulator) call each other recursively. The conceptual planner determines the salience of the concepts to be uttered and sends a stream of propositions to the linguistic module. The latter finds where the surface structure of the sentence can be extended with a formulation of the new proposition, finds appropriate words and phrases in the lexicon, and extends the output word stream. Predefined *attachment points* in the surface structure determine where and how it can be extended. These extensions seem to be limited to downward expansions and possibly coordination: "another adjective added to a certain noun phrase, a temporal adjunct added to a clause..." (1985a:189).

McDonald & Pustejovsky (1985a, 1985b) point out that there is a similarity between their 'attachment' and the grammar formalism in Tree Adjoining Grammars (TAGs; Vijay-Shankar & Joshi, 1985; Joshi, 1987). This suggests that TAGs are formalisms which may be especially suitable for incremental generation. They seem capable of simulating a variety of expansions, although the integration with other modules involved in sentence generation remains to be worked out. McDonald & Pustejovsky's (1985b) discussion of TAGs is limited to insertions. The example they work out concerns the expansion of (1a) to (1b). In a more fully incremental treatment, (1b) would be realized as a self-correction (1c), once the initial sequence (1a) has been uttered. However, McDonald & Pustejovsky's system does not seem to start uttering a sentence until it is complete, thereby obviating the need for self-corrections.

(1) a. The ships were hit.

b. The ships were reported to be hit.

c. The ships were hit ...uh... were reported to be hit.

In the KAMP system (Appelt, 1983, 1985), there is a component called TELEGRAM which couples the processes of conceptualization and formulation in an incremental architecture based on *unification*. In Functional Unification Grammar (FUG; Kay, 1979), a sentence can be produced by the *unification* of two *functional descriptions* (FDs). One of these represents a partially specified utterance and possibly includes some conceptual information. The other one is the grammar of the language. Instead of doing a single unification between a completely specified FD for the sentence as a whole (the 'text FD') and the grammar (the 'grammar FD'), the TELEGRAM planner works by gradual refinement. Initially, a high-level, incomplete text FD is produced by the planner and unified with the grammar FD. Subsequent planning produces more FDs, which are unified with the grammar FD and incorporated into the text FD. However, the system plans hierarchically, and the resulting enrichments of the text FD seem to be limited to downward expansion and possibly coordination.

There seem to be no natural language generation systems which produce incrementally in such a way that every now and then the system 'talks itself into a corner' and has to backtrack for a self-correction. Existing systems are only partially incremental: Even if they allow the conceptual input to be modified while syntactic sentence construction is already under way, the uttering of the sentence is delayed until its surface structure is complete. Thus the need for self-corrections is avoided. Moreover, there are probably no natural language generation systems in which lexical and word order choices may be the result of timing factors concerning the input to the Formulator and its processing load.

In theoretical linguistics, formal grammars seem to be biased toward one or the other kind of expansion, upward or downward. While phrase structure grammars present rules in a manner which is suitable only for downward expansion, categorial grammars specify rules for upward expansion only. TAGs seem to suffer less from this bias because they use insertion as a basic mechanism. However, TAGs only allow the addition of subtrees which are 'complete' in the sense that no sister nodes can be added.

I conclude that a new type of grammar is needed which can generate not only complete grammatical sentences and their structural trees but also sequences of incomplete trees which may arise during the planning of a full sentence.

2.3 Applications of incremental sentence generation

One could argue that there is no practical need for artificial language generation systems which can generate truly incrementally and that the risk of an occasional self-correction is only a nuisance. As long as the systems generate printed output, this arguments seems fair. But in the case of spoken output, the situation is different. Human listeners hardly have any trouble with corrections and retracings in speech. Therefore, in order to prevent unnaturally long pauses between successive sentences, the system could resort to an incremental generation strategy. An incremental parallel formulator could profitably be applied to most applications involving speech output.

Three specific applications where incremental generation is especially useful are discussed.

2.3.1 Simultaneous interpretation

Human interpreters can simultaneously parse a sentence and generate its translation. A computer model for simultaneous interpretation must therefore be radically different from conventional machine translation systems. Unlike most architectures for machine translation, which invoke the generation process only after the parse of a sentence is completed, a model for simultaneous interpretation generates the target language sentence during the parsing of the source language sentence. As soon as a fragment of the input utterance is syntactically parsed and semantically interpreted, it is passed on to the formulator, which generates the translation of that fragment while the rest of the input is parsed. The entire process, which is shown in Figure 9.2, is a variant of the IPF architecture.

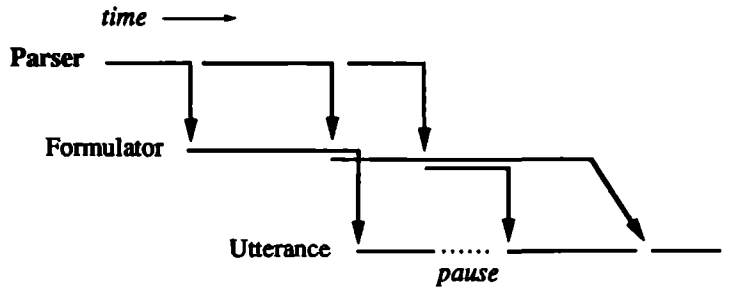


Figure 9.2
Model of simultaneous interpretation

Kitano (1989) describes a massively parallel model for interpreting telephone conversations which is intended to simulate a simultaneous interpreter at work. It is concurrent in the sense that the generation process does not wait until the entire parse is completed so that the translation is uttered incrementally. Lexical selection and partial generation are conducted while parsing is in progress. While there are some differences with respect to formalisms between Kitano's model and IPF, the general architecture is similar, and Kitano points out some difficulties which are probably inherent to any simultaneous interpreting model. Kitano's model sometimes produces utterances which are perfectly legitimate Japanese, but not the best as far as style is concerned. This is due to the fact that the optimal style in a given discourse situation sometimes cannot be determined until all or most of the sentence is parsed. This seems to be an inherent problem in simultaneous interpretation of spoken language, whether performed by humans or by computers. Kitano therefore made an option to generate a translation after a whole clause or sentence is parsed. But then the point of *incremental* generation is lost. As an alternative, I propose an interpreter which is enhanced with the possibility of retracings and restarts. Such a system could, e.g., rephrase some of the output it has just constructed to render it in a better style.

2.3.2 Multi-modal user interfaces

Some kinds of knowledge, such as structures and shapes, are hard to describe in a natural language but can be communicated to the user in an almost ideal way by means of graphical output. Icons, maps and charts are examples of graphics which are increasingly used in present-day man-computer interaction. But other kinds of knowledge, such as abstract concepts or imaginary objects, cannot be expressed so easily in a graphical way. Meanings which involve quantification, modality and negation are even more difficult to express graphically. For these kinds of knowledge, it is better to use a linguistic mode of interaction which can be readily understood by human users. Moreover, a graphical mode of interaction misses natural ways of expressing the *intention* of the communication. If a picture of an exploding nuclear reactor appears on the screen, it is not immediately apparent whether this is meant as a plain description or as a comment, a warning, a suggestion, etc. In contrast, these *illocutionary acts* can be conveyed by natural languages in many different ways, ranging from stress and intonation to word selection and the use of marked syntactic structures.

I do not argue that a linguistic output mode is better than a graphical mode, but merely that the two are complementary. A *multi-modal* system is one where a graphical and a linguistic mode of communication coexist (e.g. Bos, Claassen & Huls, forthcoming; De Smedt, Geurts & Desain, 1987; Reithinger, 1987). Such a system could be based on direct manipulation on a graphical screen, supplemented, when needed, by natural language output. Linguistic output could thus serve to clarify the graphical presentations or present a different perspective on the state of affairs. E.g., a nuclear reactor icon could be supplemented by a text explaining that there is imminent danger, etc. A *structural* view of an organization by means of graphics could be supplemented by a *functional* view in natural language. The graphical and the linguistic views can even be coordinated in an animated picture.

In several kinds of applications, natural language output seems to be almost indispensable, even when a strong graphical interaction can also be realized:

1 In *graphical programming*, e.g., a program is constructed in a piecemeal fashion by the direct manipulation of icons. However, when icons are composed in more and more complex structures, the gap from the pictorial primitives to their underlying predefined functionality widens. In such cases, natural language production may be used to explain a graphically defined program, to generate documentation, to give compiler warnings and error messages, or to provide hints and suggestions. Because icons are manipulated one by one in time, an incremental generation strategy is useful to give feedback as soon as possible (even before a complex action is completed) and can thus reduce errors.

2 Another area which I have in mind is that of *explanation in expert systems*. Most domains to which expert systems are applied involve structural knowledge (what, where?) as well as functional knowledge (how, why?). While structural aspects of a knowledge base are best presented graphically, functional knowledge is much easier to express in a linguistic fashion. In particular, natural language production can be used to

explain causal relations in the knowledge base or to clarify a particular line of reasoning to the user. Since reasoning in an expert system takes time, incremental generation might very well operate in parallel with reasoning.

2.3.3 Language description of dynamic events

In situations where the state of affairs is continually changing, e.g., when an expert system is monitoring a certain process, it is desirable to provide the user with natural language output which corresponds as closely as possible to the actual situation. Working with such a system should feel as if an expert is looking over your shoulder at the screen and explaining what's really going on. Also, when the user is specifying actions by means of direct manipulation, it is more profitable to get immediate linguistic feedback, even in the form of partial sentences, than to wait until a complete action is taken. In short, I envisage a system which provides the user with a graphical interface augmented by natural language feedback in the form of commentary, hints, warnings, etc. Such a system will have to work in a way much like the human sentence production mechanism. Immediate and continuous feedback allows early warnings because the language output provides a different and often more informative perspective on what is specified in a graphical way.

This mode of feedback calls for a mode of natural language production which is much like human sentence production. A *continuous feedback generator* should be maximally incremental, in other words, it tries to utter partial sentences as soon as possible. The pieces of information to be presented to the user by a computer system are not always likely to correspond to exactly one sentence. Sometimes more than one sentence will have to be produced, but sometimes less, perhaps only a word or phrase. Therefore, an incremental mode of sentence production is necessary. Moreover, a system providing natural language output in a situation where a state of affairs changes dynamically may be forced to backtrack and correct its own output. In order to minimize revisions and be as natural as possible, such a system should use self-corrections and other special constructions in the same way as human speakers do.

A special area where natural language output is used for immediate monitoring is the generation of a *simultaneous description* of events which are *automatically visually interpreted.* Herzog et al. (1989) describe the combination of a vision system and a natural language generation system such that image interpretation and the linguistic description of this interpretation proceed incrementally. A practical application that they propose is the description of traffic scenes. André, Herzog and Rist (1989, 1988) have designed and implemented SOCCER, a system which interprets real world image sequences and simultaneously generates natural language descriptions. The domain to which their system is currently applied is that of soccer games: in analogy to radio reports of such events, the system incrementally generates a German description of the game which it is visually interpreting but which the listener cannot see. Many other situations could be imagined where a commentary on events going on in the world is useful.

3 Segment Grammar: a unification-based formalism for incremental grammatical encoding[1]

Incremental sentence generation imposes special constraints on the representation of the grammar and the design of the Grammatical Encoder. Kempen (1987) mentions three requirements for a grammar formalism:

1 Because it cannot be assumed that conceptual fragments which are input to the Formulator are chronologically ordered in a particular way, it must be possible to expand syntactic structures upward as well as downward[2] (see also Chapter 1).
2 Because the size of each conceptual fragment is not guaranteed to cover a full clause or even a full phrase, it must be possible to attach individual branches to existing syntactic structures. This includes the incremental addition of sister nodes.
3 Because the chronological order in which conceptual fragments are attached to the syntactic structure does not necessarily correspond to the linear precedence in the resulting utterance, the language generation process should exploit variations in word order as they are made necessary by the partial utterance, while observing language-specific restrictions on word order.

Kempen (1987) proposes a unification-based formalism which meets these requirements. The present chapter attempts to further his proposal. It explains the role of unification in SG and presents a formal definition of SG as well as an informal synopsis. Finally, SG is compared with other similar present-day grammar formalisms.

3.1 Unification

Unification is proposed here as a universal information combining mechanism which is responsible for the creation of syntactic structures. A unification-based approach to grammatical encoding is useful for an incremental generator because it does not necessarily involve whole sentences. Incremental processing can be achieved by subsequent unifications. Unification-based grammatical encoding is compatible with parallel processing if unification is embedded in the necessary concurrent programming

---

[1] An earlier version of this chapter, coauthored by Gerard Kempen, has been accepted for publication in the *Proceedings of the Fourth International Workshop on Natural Language Generation*, Santa Catalina Island, 1988.

[2] Kempen presupposes the necessity of a third operation, *insertion*. However, this operation would involve undoing previous expansions, which is computationally expensive and might therefore be less desirable. Insertion can perhaps better be treated as a reformulation (due to conceptual replacement) where a substructure is possibly reused.

constructs (cf. Section 9.3). This section will describe the unification mechanism used in SG.

3.1.1 Features and agreement

Languages like English or Dutch exhibit certain regularities which are traditionally described as *agreement* (subject-verb agreement) or *concord* (gender concord). A context free grammar or phrase structure grammar (PSG) fails to capture the regularities underlying these phenomena. E.g., consider the following fragment for English:

(1) a. S → {NPsg Vsg (NP) (NP),
NPpl Vpl (NP) (NP)}

b. NP → {NPsg, NPpl}

c. NPsg → (DETsg) Adj* Nsg

d. NPpl → (DETpl) Adj* Npl

e. Nsg → chair, computer, ...

f. Npl → chairs, computers, ...

The rules for singular phrases in this grammar have nothing in common with those for plural phrases. PSGs fail to capture the fact that singular and plural phrases show essentially the same gross internal structure. There are several solutions for this deficiency. Many modern theories factor out *features*, such as number, gender and person, and formulate agreement principles as a separate part of the grammar. Thus, the arrangement of the agreeing elements is described in one part of the theory, and the details of how the elements agree are described in another part. Features are essential in unification-based grammars.

3.1.2 Unification of feature structures

Unification as a general mechanism for language generation as well as parsing was proposed by Kay (1979). In his Functional Grammar, later called Unification Grammar, now Functional Unification Grammar (FUG; Kay, 1985), generalized feature/value graphs called *feature structures* (or functional structures, or functional descriptions) are the sole informational entities. Language processing is seen as the unification of an initial feature structure with a grammar which is a large feature structure containing many alternative feature structures representing grammar rules. In unification-based grammars, a feature structure is a partial function from features to their values. E.g., there might be a function mapping the feature *plural* onto the value '-' and mapping *person* onto *1*. The notation used by Kay is:

(2)
$$\begin{bmatrix} \text{plural} = & - \\ \text{person} = & 1 \end{bmatrix}$$

As general graph structures, feature structures are recursive in the sense that feature values may themselves be structured, e.g.:

(3)
$$\begin{bmatrix} \text{cat} = & \text{NP} \\ \text{agreement} = & \begin{bmatrix} \text{plural} = & - \\ \text{person} = & 1 \end{bmatrix} \end{bmatrix}$$

A feature structure in which features share one value is said to be *reentrant*. In that case, the common value is written just once and features are coindexed to indicate the sharing relation. E.g., in the following structure, the agreement of the subject is the same as the agreement of the predicate:

(4)
$$\begin{bmatrix} \text{cat} = & \text{S} \\ \text{subject} = & \begin{bmatrix} \text{cat} = & \text{NP} \\ \text{agreement} = & \boxed{1} \begin{bmatrix} \text{plural} = & - \\ \text{person} = & 1 \end{bmatrix} \end{bmatrix} \\ \text{predicate} = & \begin{bmatrix} \text{cat} = & \text{VP} \\ \text{agreement} = & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Sharing can also be written as an equation, e.g.:

(5) <subject agreement> = <predicate agreement> (Karttunen, 1984)

(6) VP: (↑ subject agreement) = (↓ agreement) (LFG)

Unification is based on the notion of *subsumption*—an ordering on feature structures expressing their compatibility and relative specificity. Shieber provides the following definition:

> "Viewed intuitively, then, a feature structure D subsumes a feature structure D' (notated $D \sqsubseteq D'$) if D contains a subset of the information in D'. More precisely, a complex feature structure D subsumes a complex feature structure D' if and only if $D(l) \sqsubseteq D'(l)$ for all $l \in dom(D)$ and $D'(p) = D'(q)$ for all paths p and q such that $D(p) = D(q)$. An atomic feature structure neither subsumes nor is subsumed by a different atomic feature structure." (Shieber, 1986:15)

E.g., feature structure (7) subsumes (2): it carries less information and contains no differing or conflicting information. Feature structures may not be in a subsumption relation with each other because they have differing but compatible feature values, e.g. (7) and (8), or because they contain conflicting information, e.g. (7) and (9).

(7) [plural = -]

(8) [person = 1]

(9) [plural = +]

In the case of differing but compatible information, there exists a more specific structure that is subsumed by both feature structures. E.g., (2) is the most specific structure which is subsumed by both (7) and (8). In other words, the unification D combines the information from two feature structures D' and D''. This is notated as $D = D' \sqcup D''$. For a more comprehensive overview of the formal foundations of unification and its use in present-day grammar formalisms, the reader is referred to Pollard and Sag

(1987) and Shieber (1986); the latter also discusses Definite Clause Grammar as a kind of unification-based grammar.

3.1.3 Unification in Segment Grammar

In SG, the following extensions c.q. restrictions of the unification operation are proposed:

1 *Disjunctive values.* While the values of features can only be atoms or feature structures in many unification-based formalisms, SG—like FUG—allows features to contain sets representing disjunctive values. Unification computes the intersection of two value sets and succeeds if the result is not empty[3]. E.g., the unification of (10) and (11) will be (12).

(10) $\begin{bmatrix}\text{plural} = (+\ -) \\ \text{person} = (2\ 3)\end{bmatrix}$

(11) $\begin{bmatrix}\text{plural} = + \\ \text{person} = (1\ 2\ 3)\end{bmatrix}$

(12) $\begin{bmatrix}\text{plural} = + \\ \text{person} = (2\ 3)\end{bmatrix}$

2 *Non-recursive feature structures.* In many unification-based grammars there is no distinction between grammatical functions and 'true' features; thus feature values can themselves be feature structures, e.g. in (4). This also holds for a structured feature like *agreement* in (3) and (4). Recursive feature structures lead to recursiveness in the unification operation. In SG, grammatical functions are not represented as features but rather as *arcs* in segments (see below). Furthermore, SG does not use a structured feature *agreement* but specifies features for agreement individually. Hence unification is in principle restricted to a non-recursive operation[4].

3 *Non-disjunctive structures.* FUG takes seriously the idea of feature structures as the only stores of linguistic information, semantic as well as syntactic, and does not represent grammar rules other than in the form of feature structures. Since the parts of a feature structure operate conjunctively, the grammar must therefore also incorporate the notion of disjunction (or alternation). An FUG grammar is thus a large, disjunctive set of feature structures. SG does not incorporate this notion of disjunction, and indeed does not represent a grammar itself as a functional description but merely as a set of individual segments. Unification in SG is a local operation on nodes. The term *f-structure* (see below) in SG may therefore be somewhat confusing, for f-structures are

---

[3] Atomic values are considered as singleton sets; sets are written in list notation.

[4] In the present implementation, recursiveness is nevertheless possible. It is even so that features are *themselves* implemented as CommonORBIT objects, and are thus viewed as feature structures (see Chapter 8), but this recursiveness goes only one level; it is inessential and could be avoided altogether if necessary. Thus feature values can be restricted to symbols (atomic elements) or sets of symbols, which means that unification in SG is a considerably simpler operation than in most other unification-based grammars. In a similar way, Proudian & Pollard (1985) do not use general graph structures but propose restricting HPSG structures to 'flat' feature matrices where each value is a set of atomic features. When such structures are represented as vectors of integers, they can be unified in a very fast way using the 'logical AND' machine instruction.

not unified in the way feature structures (or functional structures, or functional descriptions) are unified in FUG.

3.2 Informal synopsis of Segment Grammar

Segments are the elementary building blocks of the grammar. They are graphs with two nodes: a root node and a foot node. Isolated segments are conventionally represented in vertical orientation with the root node, labeled with its category, at the top, the foot node, labeled with its category, at the bottom, and an arc, represented as a vertically directed edge labeled with a grammatical function, between the nodes. An example is shown in Figure 3.1. In running text, segments are also written left-to-right (root-to-foot), e.g. S-subject-NP or NP-head-NOUN.

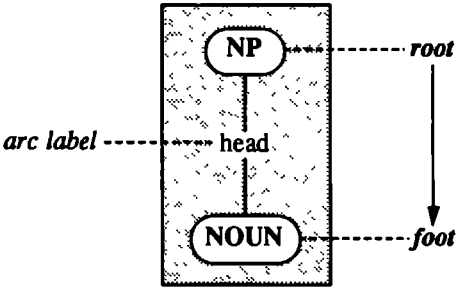


Figure 3.1
A syntactic segment

Syntactic segments are the smallest possible f-structures which express a grammatical relation and may therefore be considered as atomic units. Just like atoms in chemistry combine to form molecules, segments combine to form larger f-structures. These structures are *unordered* (they are sometimes called *mobiles*), since word order is assigned at a later stage. F-structures are graphs consisting of nodes labeled with syntactic categories (or with lexical items) and arcs labeled with grammatical relations. C-structures are ordered graphs derived from f-structures by a process described later, and phonetic strings are the sequences of terminal nodes in c-structures.

One basic operation, *unification*, governs the composition of smaller f-structures into larger ones. By unifying two nodes belonging to different f-structures, the nodes may merge into one; thus a graph of interconnected segments is formed. The two basic variants of unification are *concatenation* (vertical composition by unifying a root and a foot) and *furcation* (horizontal composition by unifying two roots). E.g., two segments which are instances of S-subject-NP and of NP-head-NOUN can be concatenated by unifying their NP nodes; two segments which are instances of NP-determiner-ARTICLE and NP-head-NOUN can be furcated, also by unification of their NP nodes. This is schematically represented in Figure 3.2.

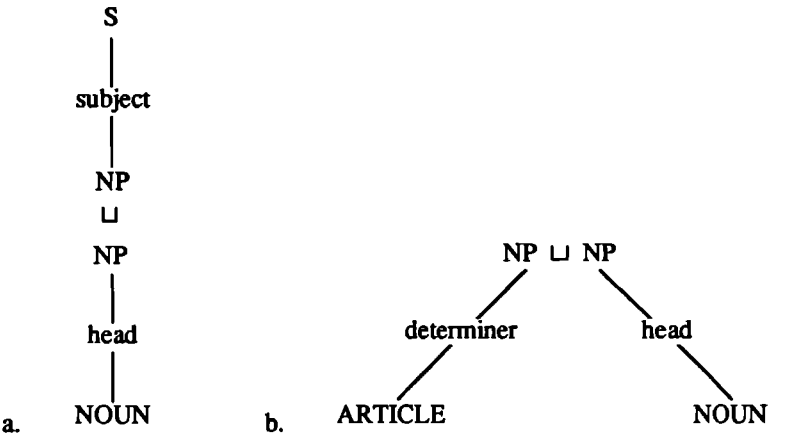


Figure 3.2
Concatenation (left) and furcation (right)

3.2.1 Features values as constraints

To each node, a set of *features* may be attributed. For example, NP nodes have a feature *nominative* with '+' or '-' as possible values. Some syntactic constraints can be rendered by specifying a feature on either the root or the foot of a certain segment. E.g., the constraint that the subject NP of an S must be nominative is expressed by setting the feature value of *nominative* on the foot node of the S-subject-NP segment to the value '+'. This is graphically shown in Figure 3.3.

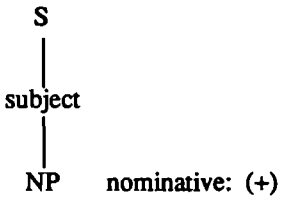


Figure 3.3
A feature as a constraint

If no values are explicitly specified, then the feature has all possible values by default, in this case the set[5] (+ -). When two nodes are unified (in either concatenation or furcation), their features are merged. This process, illustrated in Figure 3.4, is already discussed in Section 3.1. It consists of computing the union of all features in both nodes, and for each feature the intersection of the values in both nodes.

[5] Sets are represented here in list notation.

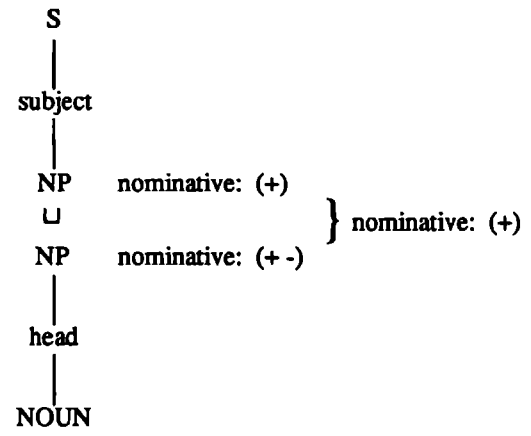


Figure 3.4
Features are merged during unification; the intersection of feature values is computed

Some syntactic constraints can be modeled by the *cooccurrence* of some feature values on the root with some feature values on the foot. Such specifications may only be given at the level of individual segments, and not at arbitrary distances, as other unification-based formalisms might allow. In Dutch, e.g., there is a constraint that an indefinite singular neuter NP has an uninflected premodifier AP. This can be represented in SG by a cooccurrence of features, as graphically represented in Figure 3.5.

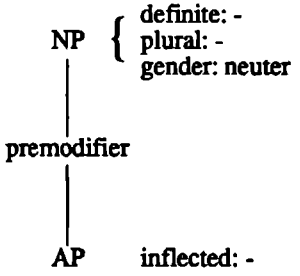


Figure 3.5
Feature cooccurrence on root and foot

3.2.2 Feature sharing, feature transport and agreement

A specification of agreement can be achieved by having the root and the foot of a segment *share* certain features, whatever their values may be. For example, the feature *nominative* is shared in the NP-head-NOUN segment as depicted in Figure 3.6.

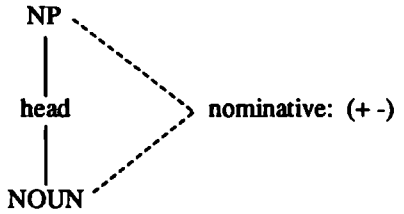


Figure 3.6
Feature sharing

Specifying shared features in a segment thus serves the same purpose as coindexing to indicate structure sharing in other unification-based formalisms. Any subsequent changes to the feature in either the foot or the root will also affect the other. An assertion that *all* features must be shared between root and foot in some segments would be a possible implementation of the Head Feature Principle (HFP) in HPSG. It is not possible to specify feature sharing between sister nodes in SG. Such agreement relations can only be indirectly enforced by specifying several individual sharing relations which are 'chained' over furcated and concatenated segments.

The combination of feature sharing and unification amounts to 'feature transport'. By virtue of the sharing relationship in NP-head-NOUN, the concatenation depicted in Figure 3.7 results in a feature change to the foot of the lower segment as well as to its root. Features are in fact not transported from one segment to another, but they are unified so that they are shared by nodes of different segments. In the case of multiple concatenations and/or furcations, features may well be shared by three or more nodes in the syntactic structure.

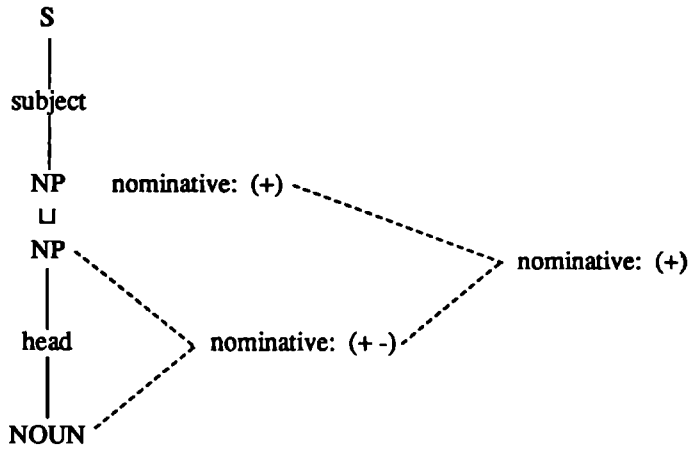


Figure 3.7
Unification of a shared feature

Agreement can easily be modeled by feature sharing in concatenated and furcated segments. For example, the features *number* and *person* are shared in S-subject-NP as well as in S-head-FINITE-VERB. If such segments are furcated by unifying their S nodes, the shared features in both segments are unified, as depicted in Figure 3.8.

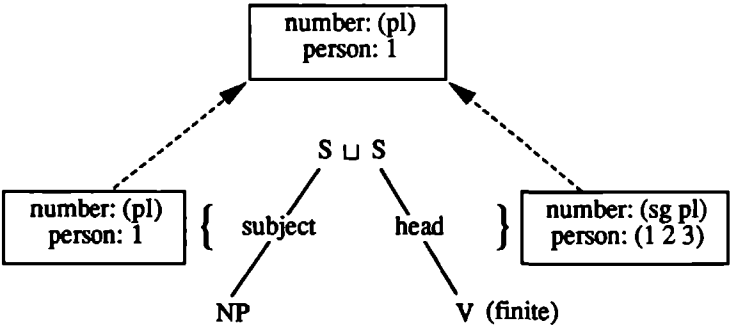


Figure 3.8
Agreement by means of feature sharing

Agreement in many unification-based formalisms can be specified between constituents at arbitrary distances by using *paths*, expressed as the coindexing in (5), or as the equations (6) and (7). The SG approach is different by virtue of the fact that agreement can only be specified on the level of individual segments, i.e., as shared features between root and foot. SG *distributes* grammatical constraints among the various syntactic segments which occur in a language. In general, an SG expresses information at the level of the segment, i.e., encapsulated in many separate objects rather than in one large structure or rule base. This makes unification more local and facilitates parallel unification (see Chapter 9).

3.2.3 Immediate dominance, sisterhood, and linear precedence

The combination of root and foot in a segment is a declarative representation of a single immediate dominance (ID) relationship. Restrictions on *sisterhood* must thus be encoded outside the segment. The addition of function words, e.g. determiners and auxiliaries (under NP resp. S nodes), is governed by *functorization* during formulation (see Chapter 4). The addition of non-function-words, i.e. constituents which are in a *case* relation to the phrase, is driven by the conceptual module but is restricted by valence information in *lexical segments* (see Section 4.3). E.g., the possible addition of a direct object in a clause is specified in lexical segments of the type S-head-V.

It is unclear whether there is a need for an explicit specification of additional, more global restrictions on sisterhood, e.g., the restriction that only one direct object may occur in a clause (cf. Starosta, 1978). I assume that conceptual input to the Formulator cannot normally give rise to such circumstances, because there will be no contradictory information in case relationships. Hence, these restrictions are seen as emerging properties of the language generation process as a whole rather than defining properties

of the grammar. If there is evidence that this kind of restrictions must be explicitly defined in the grammar, then it remains possible to specify ad-hoc restrictions on unification based on the grammatical function labels in segments. If not, then the notion of grammatical function, which was regarded as essential by Kempen (1987) may turn out to be disposable in SG.

Somewhat like the ID/LP format for PS rules, SG handles ID and linear precedence (LP) separately. LP is encoded by assigning to each foot node a feature '*positions*' which contains a list of possible positions that the node may occupy in its destination, i.e., (by default) its mother node in the f-structure. But whereas a PS-based system specifies a relative ordering of sister nodes, SG assigns a position to a constituent *independently* of its sisters; therefore, a system of *absolute* positions is used. Second, the assignment of LP may be attended with a revision of ID relations as they are specified in the f-structure. Consequently, the ID relations in the f-structure and the corresponding c-structure for a sentence may not be isomorphic. The process of assigning left-to-right order in c-structures will be further explained in Chapter 4.

3.3 Formal definition of Segment Grammar

Somewhat like a lexical-functional grammar (LFG; Kaplan & Bresnan, 1982), an SG assigns two distinct descriptions to every sentence of the language which it generates. The constituent structure (or *c-structure*) of a sentence is a conventional phrase structure, which is represented as an ordered tree-shaped graph. It indicates the 'surface' grouping and ordering of words and phrases in a sentence. The functional structure (or *f-structure*) provides a more detailed representation of grammatical relations between words and phrases, as traditionally expressed by subject, direct object, etc. The representation in f-structures also accounts for phenomena like agreement, and it does so by using features like number, gender, etc. When SG is activated in an incremental processing mode, it assigns representations to partial sentences as well as to complete ones.

When an SG is used for generation, semantic and discourse information is mapped into f-structures, which in turn are mapped onto c-structures. C-structures are then subjected to morpho-phonological processing; the resulting phonetic strings are eventually uttered as speech sounds.

A *Segment Grammar* G for a language L_G is a septuple G=(N,T,S,P,F,W,O) with N a set of non-terminal symbols, T a set of terminal symbols, S a set of segments, P a set of phonetic symbols, F a set of feature symbols, W a set of feature value symbols, and O a set of grammatical function symbols[6].

For a Segment Grammar G, *f-structures* are connected directed acyclic graphs (DAGs) defined by the quintuple (V, E, F_W, F_L, F_O) where V is a set of nodes, E a set

[6] It is questionable whether grammatical functions are strictly necessary in SG. This is discussed in Section 3.2.3.

of arcs: $E \subseteq V \times V$, F_W a partial function[7]: $F \times V \rightarrow \wp(W)$, F_L a labeling function: $V \rightarrow N \cup T$, and F_O a labelling function $E \rightarrow O$. The set of f-structures in G is defined as $\mathcal{F}_G = S \cup \{y \mid \exists\, y', y'' \in \mathcal{F}_G : y \in U(y',y'')\}$ where U is the universal unification function: $\mathcal{F}_G \times \mathcal{F}_G \rightarrow \wp(\mathcal{F}_G)$ which derives new f-structures by unifying a node of one f-structure with a node of another. Unification of nodes is introduced earlier in this chapter (Section 3.1).

An f-structure is well-formed if it observes the principles of completeness and coherence stated in (13) and (14). These principles refer to valence specifications which are projected from the lexicon.

(13) An f-structure is *coherent* if all the grammatical functions of each node are governable by that node, i.e., if $\{\forall(v,x) \in E \mid F_O(v,x) \in O_v\}$, where O_v is the set of possible grammatical functions for v.

(14) An f-structure is *complete* if each non-terminal node contains all its obligatory governable grammatical functions, i.e., if $\{\forall v \in V \mid \exists(v,x) \in E, F_O(v,x) \in O_v\}$, where O_v is the set of obligatory grammatical functions for v.

Each *segment* $s \in S$ is an f-structure with $V = \{r,f\}$ and $E = \{(r,f)\}$ where r is called the root node and f the foot node. The subset of segments $\{s \in S \mid F_L(f) \in T\}$ is called the set of *lexical segments.*

For a Segment Grammar G, *c-structures* consist of a quadruple $(V, F_M, <, F_L)$ where V is a set of nodes $\subseteq V$, F_M a mother function: $V \rightarrow V \cup \{\perp\}$, $<$ a well-ordered partial precedence relation: $< \subset V \times V$, and F_L a labeling function: $V \rightarrow N \cup T$. While for each well-formed f-structure, there is a c-structure such that: $V_c \subseteq V_f$, there is no isomorphic mapping from c-structures to f-structures. The c-structures in G are derivable from the f-structures by means of the destination and linearization processes which are described in Section 4.4.

For a Segment Grammar G, *phonetic strings* are structures $\in P^*$. The phonetic strings in L_G are computed by the Phonological Encoder from the sequences of terminal nodes of all possible c-structures in G.

3.4 Discussion and relation with other work

The use of segments for the expression of grammatical knowledge is advantageous in an incremental sentence Formulator. I will sum up some of these advantages here and at the same time draw comparisons with other work.

3.4.1 Phrase Structure Grammar

PS rules are string rewriting rules which express immediate dominance (ID, motherhood) relationships *together* with sisterhood (co-occurrence) and linear

---

[7] $\wp$ denotes the powerset. Sets of feature values, as allowed by the function F_W, represent disjunctive values.

precedence (LP) relationships. Hence, it is often necessary to express the same dominance relationship more than once, namely for every possible sisterhood and linear precedence relationship. The example in (15) is from Sells (1985).

(15) a. VP → V NP kiss the bride

b. VP → V NP PP send the message to Kim

c. VP → V NP S' tell the class that break is over

d. VP → V NP VP expect results to be forthcoming

It must be admitted that pure PSG is seldom used. In Government and Binding (GB) theory, which uses PS rules mainly to specify hierarchical structure, order is fixed by other components of the grammar, such as case assignment. Generalized Phrase Structure Grammar (GPSG; Gazdar, Klein, Pullum and Sag, 1985) uses rules in ID/LP format, where a comma on the right hand side of the rewrite rules indicates that the categories are unordered. These rules are then complemented with separate rules for precedence relations, e.g.:

(16) a. VP → V, NP kiss the bride

b. VP → V, NP, PP send the message to Kim

c. VP → V, NP, S' tell the class that break is over

d. VP → V, NP, VP expect results to be forthcoming

e. V < NP < XP

Notice that, although the linear precedence is now encoded separately, ID relations are still expressed redundantly. SG offers a more economic way of encoding ID by specifying only one relationship between a pair of nodes at a time.

But the real problem in incremental production is that the choice between rules (17a-d) cannot be made deterministically (cf. Kempen & Hoenkamp, 1987). If daughter constituents are produced one at a time by means of a PSG, the system will be forced to choose between rules and backtrack when necessary. Although in GPSG, subcategorization indices will solve the choice between rules (17a-d), this does not work when several subcategorization frames are possible for one predicate and constituents may be added by incremental upward as well as downward expansions. By virtue of its orientation toward the representation of separate ID relationships, SG allows the incremental addition of sister nodes and chooses between subcategorization frames while the construction of the syntactic structure is already under way.

This problem with PS rules could, in theory, be obviated by redefining the grammar as in (17). But then the PS structures generated by the grammar would not be isomorphic to those generated by grammar (16): the grammars are only weakly equivalent.

(17) a. VP → V, NP, VPrest

b. VPrest → Ø

c. VPrest → PP

d. VPrest → S'

e. VPrest → VP

3.4.2 Categorial Grammar

Categorial Grammar (CG) is somewhat similar to SG, because grammar knowledge is distributed among several individual entities. Whereas a SG distributes the grammar among syntactic segments, a CG represents a grammar as a set of structured categories. But although classical CG, unlike PSG, is lexically guided and therefore suited for generation, a similar objection could be raised against it. In classical CG, word order and co-occurrence constraints are encoded as syntactic types on lexical items. Whatever choices there are with respect to either LP or sisterhood will result in alternative syntactic types for the same word. For languages with relatively free word order or many sisterhood alternatives, this may result in a drastic increase of categories encoded in the lexicon. By comparison, the opposite is true for SG, where a relatively word order free language will make the grammar relatively simpler.

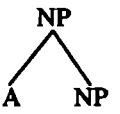
It must be noted, however, that some kinds of CG allow considerably more flexibility. By choosing non-directional categories rather than right-directional and left-directional ones, some variations in word order can easily be accounted for. Using rules such as composition and category-raising, 'flexible' CGs could even provide an elegant description of a free word order language such as Warlpiri (Bouma, 1986). Flexible CGs operate by producing many possible derivations for a given sentence rather than assigning a unique constituent structure. Since syntactic rules operate in parallel with semantic rules, CGs are reversible, and this suggests that incremental processing is facilitated not only for parsing (e.g. Bouma, 1989) but also for generation. However, flexible CGs suffer from spurious ambiguity which may present a control problem for the generator. It remains to be seen whether this problem can be solved in a way which is not at the expense of its incremental capacity.

3.4.3 Tree Adjoining Grammar

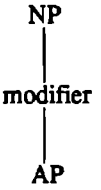
Tree Adjoining Grammar (TAG; Joshi, 1987) is a tree generating system consisting of a finite set of elementary trees and two composition operations (*substitution* and *adjoining*) which build derived trees out of elementary trees. Like SG, and as opposed to PS-rules, TAG is tree-based rather than string-based. FTAG, the recent *feature structures based* extension of TAG (Vijay-Shanker & Joshi 1988) uses unification of features as a clearer way of specifying restrictions on tree adjoining, and is therefore even more similar to SG. The role of some elementary trees in TAG is comparable to that of SG segments, while adjoining takes the role of unification. For example, the auxiliary tree for an adjective (18a) can be said to correspond to the NP-modifier-AP

segment (18b), although it must be noted that SG always creates an adjectival *phrase* rather than just an adjective.

(18) a.



b.



The difference between the two approaches is, that TAG allows the *recursive* insertion of A nodes, whereas SG allows the *iterative* addition of AP nodes. In terms of PSG, the corresponding rules are:

(19) a. NP → A NP (TAG)

b. NP → AP* N (SG)

The adjoining operation of the auxiliary tree for an adjective yields two NP nodes in the resulting structure (Figure 3.9a), whereas the corresponding composition of SG segments will result in only one NP node (Figure 3.9b). Whereas it could be argued that the hierarchical embedding of NPs in TAG allows an easier semantic interpretation in some cases (e.g. for the order of non-compositional adjectives: *an old strong man* vs. *a strong old man*) it also makes for considerable redundancy in cases where recursiveness does not affect semantic interpretation.

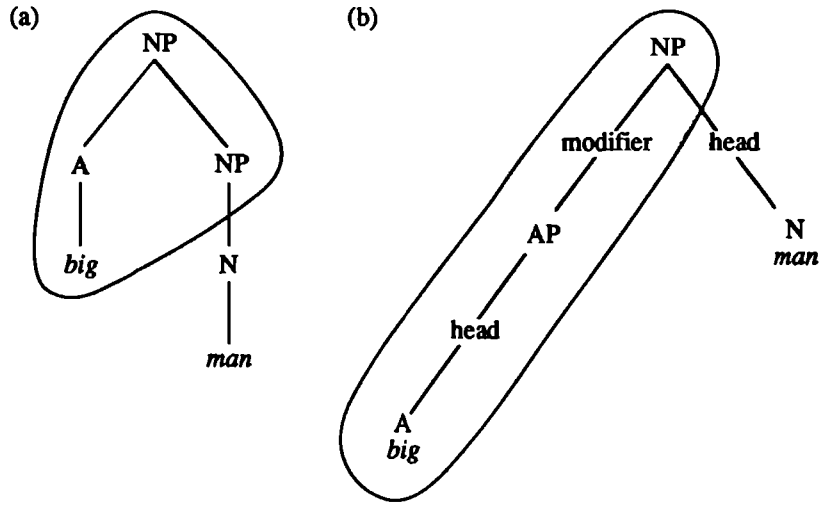


Figure 3.9
Results of TAG adjoining (left) and SG unification (right)

Word order and immediate dominance are factored in the TAG formalism, which provides considerable flexibility in the generation process of a sentence. TAG allows incremental generation, but only as defined by the adjoining operation, which means that it does not allow the addition of sister nodes *(furcation)* without structural growth in vertical direction. Unlike SG structures, TAG trees always contain all sisters. E.g., completely different elementary trees are distinguished for transitive and intransitive verbs. This is problematic for languages like English and Dutch which have many verbs that can be either intransitive or transitive. It does not seem possible to build a transitive tree incrementally by starting with an intransitive tree and expanding it into a transitive one by means of furcation, as SG might allow (if valence permits it).

Also, TAG seems to require, for many lexical items, a number of variants of elementary trees. For example, the transitive verb *eat* requires separate elementary trees for the constructions *Subj-V-Obj, WH(Obj)-Subj-V, to-V(infinitive)-Obj,* etc. Since the speaker presumably has to choose between such alternatives at an early stage, a TAG-based generation model faces choice problems which are avoided by SG. There, the choice between such constructions can be postponed to a later stage, so that minimal commitments with respect to further incrementation are imposed.

3.4.4 Lexical-Functional Grammar

With respect to the representation of linguistic structures, SG is perhaps closest to Lexical-Functional Grammar (LFG; Bresnan, 1982). Both formalisms assign an f-structure as well as a—not necessarily isomorphic—c-structure to a sentence. This distinction is felt to be a useful abstraction in both formalisms, and is used to account for such things as functional control (e.g. S-O raising, cf. Chapter 5). The structures which are assigned by both formalisms to sentences are often very similar. S-rules in LFG which are annotated with functional schemata contain essentially the same information as syntactic segments. A basic LFG rule such as (20a) could in principle be translated into two SG segments (20b)[8].

(20) a. $S \rightarrow$ NP VP
$\uparrow \text{subj} = \downarrow \quad \uparrow = \downarrow$

b. S-subject-NP, positions: (1)
S-head-VP, positions: (2)

However, upon closer inspection, a number of differences become apparent. Whereas in LFG, c-structures are generated independently by means of a PSG, SG derives c-structures directly from f-structures (this is explained further in Chapter 4). In LFG, which forbids operations on f-structures, there is no way to derive a c-structure from an f-structure, although derivation can be f-structure driven (Wedekind, 1988).

LFG encodes functional relations as well as features uniformly in f-structures; functional structures are thus feature structures. In SG, however, only *nodes* are represented as feature structures; f-structures form a different graph. The advantage of

[8] This was suggested by Gosse Bouma.

the latter representation is that unification is a more local operation and can be simpler because it need not be recursive.

3.5 Concluding remarks

SG describes sentences of a language in terms of syntactic segments—atomic units larger than syntactic nodes—and their possible combinations, governed by unification. Because SG specifies ID relations at the level of individual segments, f-structures can be generated by adding daughter, mother, *and* sister nodes incrementally. Because SG specifies LP relations at the level of individual segments, word order can be determined for partial sentences. These properties make SG a particularly flexible formalism for incremental generation. Although other formalisms can in principle be adapted for incremental generation, a lot of bookkeeping would be required to achieve the same effect (e.g. backtracking in PSGs).

In the form of features and word order information, segments contain all necessary information which is traditionally associated with grammatical functions such as subject, direct object, etc. Although grammatical function is coded as arc labels, this is done for clarity, and not a single syntactic decision in the Formulator need rely on these arc labels as such.

The choice of unification as a mechanism for the composition of syntactic informational structures out of segments is motivated by the fact that unification is a general, adequately formalized formalism. A wide range of present-day linguistic theories is based on unification, which allows their explicit construction, computer implementation and mutual comparison.

Although other 'contemporary' grammar formalisms (e.g. GPSG, CG, TAG, LFG) offer considerable flexibility for incremental parsing and generation, SG distinguishes itself from these formalisms mainly in that it views the segment as a domain of locality. It seems that syntactic segments, as they are proposed here, are elementary structures which are small enough to allow any kind of incrementation, yet large enough to hold information (e.g. features for agreement, valence, LP rules) without having to resort to any additional global syntactic rules outside the segment definitions. I believe that this approach provides the grammar with more modularity because the set of segments for a language is easily extensible and modifiable. I do not want to extol on the potential linguistic coverage of SG but to clarify some concepts in the formalism I will present a typical SG treatment of some syntactic constructions in Chapter 5.

I conclude that SG is not proposed as another formalism which is claimed to be formally more powerful or less powerful than other grammars, or accounts for different linguistic facts. Rather, it is a grammar where linguistic knowledge is organized so as to *serve* a specific language *processing* model. This model will be the subject of Chapter 4.

4 Syntactic structures in IPF[1]

IPF (Incremental Parallel Formulator) is a conceptually and lexically driven generator: *syntactic segments* containing content words are chosen from the lexicon and are incrementally fitted into a syntactic structure (*f-structure*). Case relations result in the establishment of grammatical relations in the form of *intercalating* segments, while function words are added in the form of *functor* segments. Meanwhile, a *c-structure* which represents constituency and left-to-right order on the 'surface' level is incrementally derived from the f-structure. Because these structures are built in parallel by independent processes, their form is partly dependent on the order of the input as well as on the computational load of the formulation processes. IPF has been implemented and tested on a Symbolics Lisp machine. This implementation proved helpful to make many computational aspects of the model more concrete and testable.

4.1 Introduction

4.1.1 Tasks of the Grammatical Encoder

The division of the generation process into distinct modules requires a demarcation of which modules are responsible for which linguistic tasks. A schematic overview of the tasks and output of the Grammatical Encoder and the modules which it is connected to is given in Table 4.1.

Table 4.1
Tasks and output of the Grammatical Encoder and connected modules

Module	Tasks	Output
Conceptualizer	Inferences, paraphrases	Concepts, semantic roles, features
Grammatical Encoder	Lexico-syntactic choices	Words in an ordered constituent structure
Phonological Encoder	Inflection, phonology	Phonological strings in order

[1] An slightly different version of this chapter has been accepted for publication in the *Proceedings of the Second European Natural Language Generation Workshop*, Edinburgh, 1989.

It is not the task of the Grammatical Encoder to check whether the output of the Phonological Encoder is semantically well-formed. E.g., the principle of allowing just one instance of a semantic case per simple clause, which is embraced by the Lexicase theory (Starosta, 1978) is a semantic constraint rather than a syntactic one. Thus it should be handled by the Conceptualizer. Choices in the Grammatical Encoder may be conceptually driven, but they may also originate in purely syntactic constraints. E.g., the utterances (1a,b) are related by inferences in the Conceptualizer. In contrast, the choice between utterances (2a,b) may depend on timing effects which are due to an incremental mode of language generation.

(1) a. I think you shouldn't go.

b. I don't think you should go.

(2) a. Tomorrow he'll come.

b. He'll come tomorrow.

Similarly, lexical choices may be due either to conceptual decisions or to syntactic variation. E.g., the utterances (3a,b) are related by a conceptual inference, which uses semantic viewpoints to guide its decision. In contrast, decisions between different syntactic categories, e.g. in utterances (4a,b), are taken by the Formulator, which operates within the lexico-syntactic constraints imposed by incremental generation.

(3) a. John *lives* in Paris.

b. John is an *inhabitant* of Paris.

(4) a. Mary wanted John to *translate* Chomsky.

b. Mary planned John's *translation* of Chomsky.

This separation of tasks presupposes that *live* and *inhabitant* are semantically related, but not syntactically, and that *translate* and *translation* are syntactically related, but semantically equivalent[2]. IPF employs Segment Grammar (SG; see Chapter 3), which is a *lexical grammar* in the sense that the lexicon is an autonomous component governed by lexical rules. The role of the lexicon will be further explained in Section 4.3.

4.1.2 Conceptual and lexical guidance

It is an old debate, whether lexical selection or a syntactic frame is the driving force in sentence generation. Skinner (1957) considered lexical selection the fundamental process in language production. *Key responses* (lexical items) are chosen first; they are then arranged and modified by *autoclitic responses* (which include word order and inflection). Chomsky (1959) argued that the opposite is just as likely: the key

[2] It could of course be objected that *translation* can not only refer to the activity expressed by *translate*, but also to the product of this activity. The solution is then, to distinguish two semantically different lexical entries for *translation*.

responses are chosen only after the grammatical frame has been generated. Clearly, Chomsky's argument was in defense of his early TGG theory which inserts lexical items in a previously constructed syntactic structure.

In incremental sentence production, both positions are valid—to a certain extent. If syntactic structures have a conceptual origin, then the properties of lexical items rendering those concepts in the input must clearly be taken into account. This is the principle of conceptual and lexical guidance. Kempen and Hoenkamp (1987) show convincingly that the syntactic subcategorization properties of lexical items govern the choice of a syntactic frame. E.g., the similarly structured conceptual representations underlying (5a,b) result in different syntactic structures due to the differing syntactic subcategorization properties of *know* and *want*. Constructing a syntactic frame before selecting lexical items could result in ungrammaticality (5c).

(5) a. John knew he hit Peter.

b. John wanted to hit Peter.

c. *John wanted he hit Peter.

Approaches which persist in the assumption that lexical material (i.e. content words) is inserted into a previously made syntactic structure (e.g. Dell, 1986) are therefore unrealistic from a psycholinguistic viewpoint. Lexical guidance of grammatical encoding is present in many modern linguistic theories. In Government-Binding theory, it is formulated as the Projection Principle, which states that syntactic representations are 'projected' from the lexicon, in that they observe the syntactic subcategorization properties of lexical items (Chomsky, 1981).

On the other hand, lexical items to be incorporated in one sentence cannot be chosen independently of one another. If a lexical item gives rise to the construction of a syntactic frame which observes its subcategorization properties, then the constituents of this frame will have to fit into its categorial restrictions. For example, *to dedicate* will give rise to an S which may take an NP, but not a subclause, as a direct object. If, subsequently, the direct object is to be lexicalized, this categorial restriction will force a nominal lexical entry to be preferred to a verbal one. Thus, in incremental sentence production, the choice of lexical material is subject to categorial restrictions imposed by the partial tree which has been constructed so far. Incremental generation is also partially syntax-driven in this sense.

Moreover, IPF does not assume any particular order in which conceptual material is input into the Formulator. Consequently, it is very well possible that the direct object NP is created first. A subsequent upward expansion of the syntactic structure may then cause the NP to obtain a role in an S which has been created at a later stage (see Chapter 1). Thus, subcategorization restrictions do not always propagate from the top of the tree downward; in the case of upward expansion, restrictions propagate upward as well. This possibility is not widely recognized. E.g., Word Grammar (Hudson, 1984) presupposes an inherent asymmetry in the *companion* relations between words. This asymmetry extends to lexical selection: the head decides which words are selected in collocation with the head, but not the other way around (1984:78).

4.2 The generation of f-structures

The Grammatical Encoder first creates f-structures, which reflect grammatical relations between constituents. Meaning elements are encoded as syntactically categorized lexical elements, syntactic relations and features. These are incrementally added to form a syntactically coherent functional structure.

It will now be explained in detail what happens when a conceptual message enters the Formulator. Recall that there are three kinds of input messages to the Formulator: semantic concepts, semantic roles, and features. To explore incremental production in a fine-grained way, each input message is processed individually. Each input message will spawn its own parallel formulation process which may overlap with other processes.

4.2.1 Lexicalization

The first kind of message which the Formulator may receive is a request to create a linguistic expression for a semantic *concept*. For each such input, a linguistic *sign* is brought about with the semantic concept as its referent. An example input is the following request to formulate a sign referring to a concept "an apple":

```
(FORMULATE (A SIGN
              (CONCEPT (AN APPLE)))
```

This *sign* is actually nothing else but a generic, empty syntactic category, to be refined to a more specific category by a grammatical encoding process. In contrast to Steels and De Smedt (1983) who use the inheritance hierarchy to refine a linguistic sign, this kind of refinement is performed by an independent lexicalization process, which selects lexical entries and subsequently unifies the sign with one of these entries. We will not be concerned here with the semantic aspects of lexicalization, but only with the syntactic aspects.

Lexical entries in SG are not simple strings but are instead structured objects. It will not be surprising that the basic building block of the lexicon is the segment rather than the word. A *lexical segment* is a segment where the foot is a word. Content words are lexical segments where the arc is *head*. As in X-bar theory, the category of the phrasal node is determined by that of its head, e.g. NP-head-NOUN, S-head-VERB, PP-head-PREPOSITION, etc. Examples of the English lexicon are NP-head-CITY (a nominal lexical segment) and S-head-TRY (a verbal lexical segment).

The process of lexical selection can be governed by different control structures, e.g., parallel or sequential. It is likely that the human speaker accesses lexical entries in parallel (Levelt, 1989), so IPF selects an entry from competing parallel processes. However, since the sign to be refined can only unify with one entry at a time, lexicalization proceeds in a largely sequential mode anyway. If the sign to be lexicalized is unrelated to any other sign, then obviously the first refinement by unification will succeed. This is exemplified in Figure 4.1.

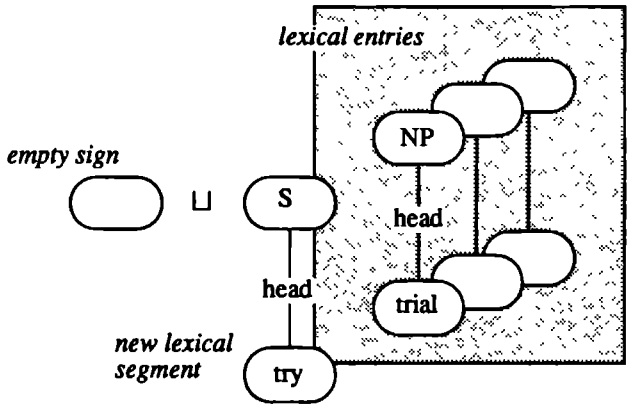


Figure 4.1
Refinement by unification

However, if the sign sign to be lexicalized is already syntactically related to another one in the current sentence, its lexicalization may be subject to subcategorization restrictions. Thus, unification may fail; if so, another lexical entry may succeed. When needed, the lexicalization process will even attempt to create new lexical entries (see Section 4.3.3). If no lexical entries succeed in unifying, a syntactic dead end is signalled (see Chapter 1) and perhaps a repair or restart occurs. Figure 4.2 illustrates this selection process.

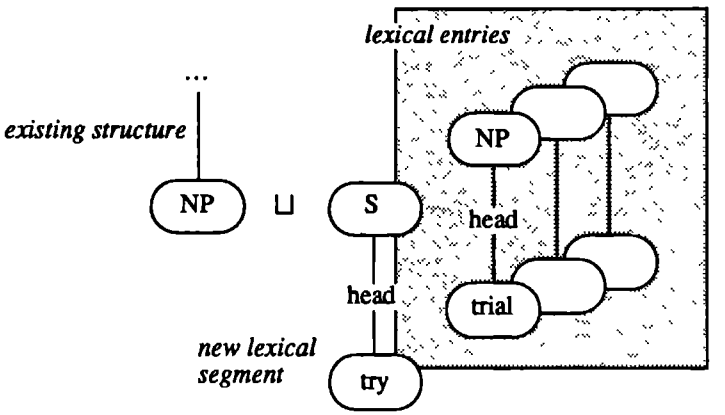


Figure 4.2
Syntactic subcategorization affects lexicalization: when the verbal lexical entry *try* has been rejected, the nominal lexical entry *trial* may unify with the existing NP

4.2.2 Case, grammatical function, and intercalation

A second kind of conceptual message which the Formulator may receive is a request to formulate a case relation between two linguistic signs. Case relations signal which

semantic roles are filled by semantic concepts in other concepts. An example input is the following request to establish an *agent* relation between two signs:

```
(LET ((SIGN1 (A SIGN
                (CONCEPT 'OTTO)))
      (SIGN2 (A SIGN
                (CONCEPT (AN EAT))))
   (DEFINE-CASE SIGN1 'AGENT :IN SIGN2))
```

The fundamental insight that the verb governs its complements forms the core of the theory of the French linguist Tesnière (1959), a progenitor of case and valence theory. The number of slots that a dependent constituent (*actant*) may occupy in a verb frame he calls *valence*. Unfortunately, Tesnière confused syntactic and semantic roles in his labeling of verb frame slots. Fillmore (1968) avoided this confusion by distinguishing a set of *cases* or universal semantic relations, e.g., *agent, patient, instrument,* etc., which belong to an underlying semantic structure. Fillmore, however, confused relational and categorial notions: his case relations were originally defined in terms not only of the perception of roles in external situations but also in terms of semantic features such as *±animate*. Later he became more careful to keep these distinct (Fillmore, 1977). In the present study, semantic case labels are considered as purely arbitrary and conventional means of distinguishing the various semantic relational slots in a linguistic sign—not only in verbal categories, but, by extension, also in nominal and other categories, in as far as they have dependent constituents.

The bridge between descriptions of situations and syntactic structure is provided by means of a *case frame* which assigns syntactic roles to particular participants in the situation represented by the sentence. Since lexical entries in SG are phrasal categories, it is logical to attach case frames to the roots of lexical segments rather than to the feet (see also Section 4.3.1). A case frame is represented as an association list, where with each case a syntactic function is associated. Sometimes more information may be associated with the function, e.g. categorial restrictions, a 'surface' case marker in the form of a preposition, and an indication of whether this function is optional (marked by a 0) or obligatory. Furthermore, a case frame includes not only 'pure cases' (Tesnière's *actants*), but also modifiers and adjuncts, i.e. Tesnière's *circonstants*). A simplified example of a case frame is the following one, which may be transitive or intransitive and has an optional modifier using the preposition *in*. It could be part of the case frame of *to eat* or Dutch *eten*.

```
((AGENT (SUBJECT))
 (THEME (DIRECT-OBJECT NP) 0)
 (LOCATION (MODIFIER IN-LEMMA) 0))
```

When the Formulator receives a request to formulate a semantic role between two linguistic signs, this role is looked up in the case frame of the superior sign. This, of course, presupposes that this constituent has already been lexicalized, for otherwise no case frame is available. If necessary, the process waits until lexicalization has occurred (see Section 9.2). The second step consists of looking up the grammatical relation associated with the semantic role. This will yield one or more segments with that

grammatical relation as arc label, e.g., S-subject-NP. These segments are called *intercalating* segments, because they relate one phrase to another. The choice between possible intercalating segments, e.g., between S-subject-NP or S-subject-S, is constrained by the category of the foot node. If more than one intercalating segment is possible, they are tried one after the other until one fits between the two phrases. The intercalation mechanism is schematically shown in Figure 4.3.

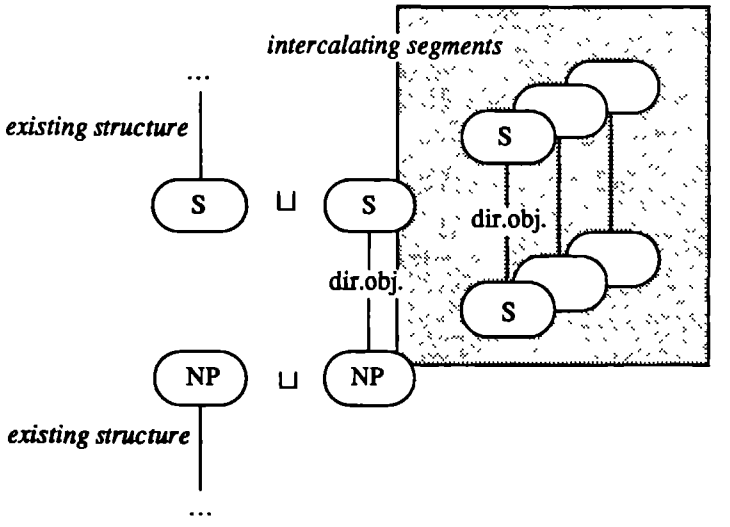


Figure 4.3
The intercalation process

If the node which is to unify with the foot of an intercalating segment (the *case filler*) has not been lexicalized yet—which is very well possible—then clearly the first intercalating segment will succeed; this must be considered a random choice. However, a subsequent lexicalization may invalidate this choice, and backtracking may be necessary. If, on the other hand, the node at the foot is lexicalized *before* an intercalating segment is chosen, then it is to late for the phrase at the root to enforce its subcategorization restrictions on the foot; there may not be a suitable intercalating segment to relate the two nodes and a repair may also be necessary. The problems associated with backtracking in IPF have not been discussed so far; a general solution will be proposed in Chapter 10.

As already mentioned, it is normally the *foot* of an intercalating segment which unifies with the case filler. However, a case frame may specify a surface case marker in the form of a preposition. When an intercalating segment gives rise to a PP at the foot of the segment, the surface case marker is attached as the *head* of the prepositional phrase. The case filler then unifies not with the PP itself but with the *object* of the PP—an NP or S. This is shown in Figure 4.4.

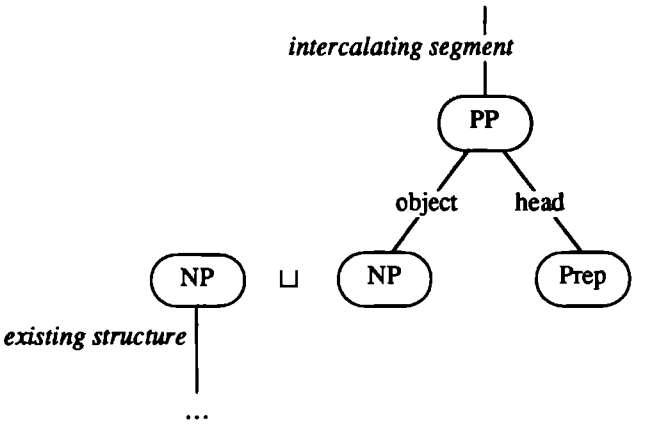


Figure 4.4
Intercalation with a PP as foot

This treatment of PPs is clearly deviant from that of other constituents. An alternative solution would consist of eliminating PPs altogether and associating surface case markers directly with the categories NP or S. Such a treatment of 'preposition-initiated' NPs is also proposed elsewhere (Dekeyser, Devriendt, Tops & Geukens, 1979). A drawback of the latter approach is that the generality of preposition-initiated constituents is lost, since in Dutch the *object* of a PP can also be an S.

4.2.3 Features and functorization

Features (e.g., *plural* or *definite*), are formulated in IPF by simply assigning them to a node. Certainly such semantico-syntactic features are derivable from other semantic or discourse information, but this will not concern us here. Thus the Formulator may receive, e.g., the following request to assign the value '-' to the feature *definite* of a sign, and the same to the feature *plural*:

```
(LET ((SIGN1 (A SIGN
                (CONCEPT (AN APPLE))))
  (DEFINE-FEATURES SIGN1 '(DEFINITE - PLURAL -)))
```

It will be assumed that this assignment of features to linguistic signs occurs *before* they are lexicalized. This is because they may actually restrict lexicalization, for they play a role in the unification with lexical segments. Assignment *after* lexicalization is not guaranteed to be successful and might require backtracking.

The main role of features is that they guide various *functorization* processes, e.g., the addition of function words or inflections. The manner in which features trigger functorization is language dependent. E.g., when a sentence in Dutch gets the value '+' for the feature *perfect*, an auxiliary is added. Other languages such as Latin, e.g., share the feature *perfect* between S and the finite verb, which undergoes an inflectional change. Features which trigger inflectional change do not play a role until the

phonological encoding stage. On the other hand, features which surface as function words will give rise to the addition of one or more segments, which are called *functor* segments. These segments are associated with categories, e.g., determiners are associated with the NP and auxiliaries are associated with the S. Unification is used as a general mechanism to choose among various possible functor segments.

In Dutch, e.g., one of several articles can be added to the NP, depending on definiteness, gender and number[3]. IPF will try to unify the roots of all these functor segments with the NP until one succeeds. In principle, this could be a parallel operation, except that the NP cannot be involved in more than one unification at the same time. The addition of an article by means of functorization in the NP is schematically represented in Figure 4.5. The constellation of features in this particular example will lead to the selection of the article '*een*'.

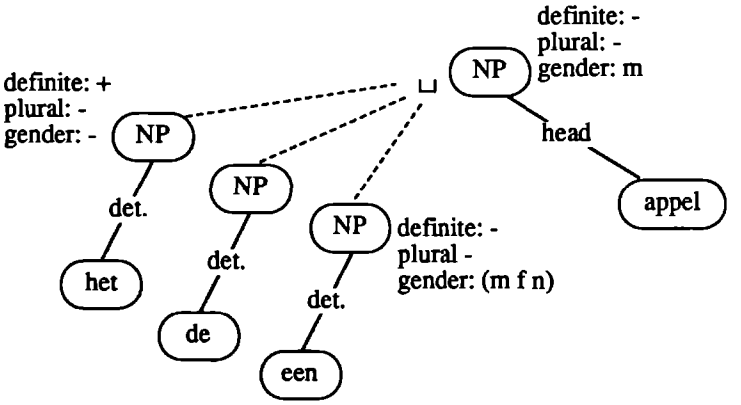


Figure 4.5
Functorization of the NP

On the S level, functorization may result in the addition of auxiliaries. This is performed in a fashion similar to functorization in the NP. In addition, it is decided which verb (main or auxiliary) shares with S the feature *finite*. A finite verb agrees with its matrix sentence, in the sense that features like *person* and *plural* are shared between them[4].

It is clear that conceptual revisions causing feature changes may result in syntactic revisions which affect functor segments, just like concept replacement may result in syntactic revisions affecting lexical segments.

---

[3] It must be added that functorization of the NP is slightly simplified here, for it deals only with articles, and does not account for quantifiers and other elements which may occur in conjunction with—or in replacement of—articles.

[4] Following a classical TGG approach, this could also be modeled as the addition of a segment with as its foot a constituent +INFL.

4.3 The structure of the lexicon in Segment Grammar

It is already indicated that *lexical segments* are the basic building blocks of an SG lexicon. This section will discuss the lexicon in more detail.

4.3.1 Lexical segments

Because lexical segments always link a word to a phrase, the lexicon is essentially a *phrasal* lexicon. The representation of lexical entries as lexical segments allows a clear separation of two kinds of information traditionally assigned to words, as schematically represented in Figure 4.6 for the Dutch entry *zien* (to see):

1 *Syntactic/semantic* information: syntactic category and valence (in the form of a subcategorization frame), as well as the meaning content of the lexical entry, are assigned to the roots of lexical segments (the phrases). This part of a lexical entry is sometimes called the *lemma* (Kempen & Huijbers, 1983).
2 *Morpho-phonological* information: morphological class associated with a categorial label (part of speech), as well as the phonological sound form, are associated with the feet of lexical segments (the words).

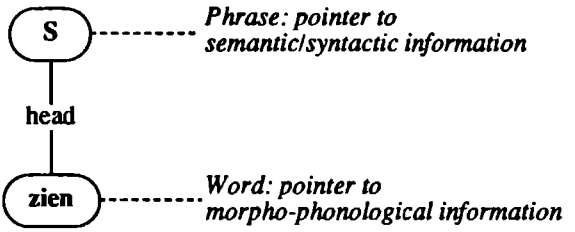


Figure 4.6
Distribution of information in a lexical segment

This separation of knowledge in terms of root (phrase) and foot (word), as shown in Figure 4.6, is advantageous in SG because knowledge is stored where it is most directly relevant. Syntactic information is always relevant to phrases, never to words. Thus, e.g., syntactic subcategorization information in a segment S-head-ZIEN (a Dutch verbal segment for *to see*) is attached to the root node of the segment, i.e. the S node. This is logical because it concerns the structure of the S, not that of the verb. In contrast, morpho-phonological information is stored in the foot node, e.g., the fact that *zien* is a strong verb and that its stem is *zie*. Clearly the latter kind of knowledge is relevant only at the word level, not the phrase level.

Another, independent argument for this separation of information can be found in homonymic relations: the same verb can sometimes function as a main verb or as an auxiliary (e.g., *to have*), or has either a transitive or intransitive subcategorization frame, often depending on its semantic frame (e.g., *to boil*). Still, in such cases, the morpho-phonological characteristics of such homonymic verbs are almost always

identical, which is a strong indication that they are indeed identical objects. Consequently, it is better to factor out the differences and assign them a separate place in the representation. This is also in line with Levelt (1989:187-188) who proposes a separation between syntactico-semantic and morpho-phonological information, the root and the foot of a lexical segment hold different, complementary kinds of knowledge:

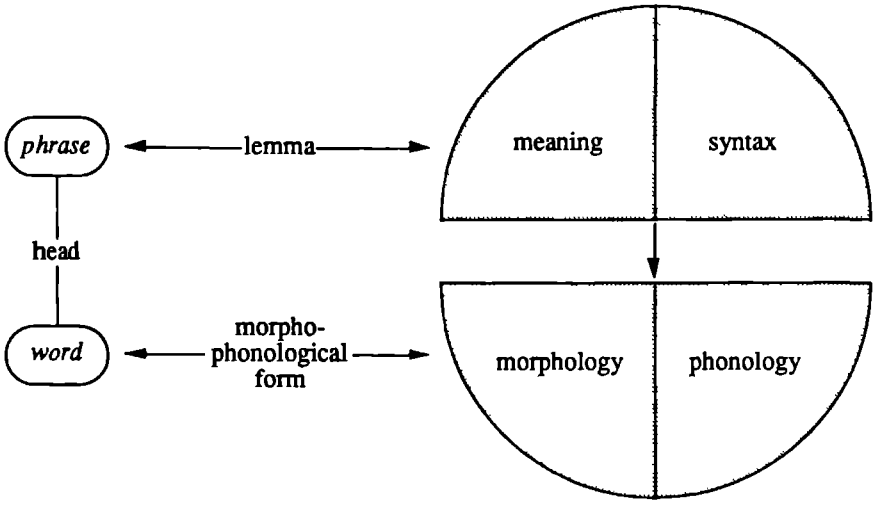


Figure 4.7
Distribution of information in a lexical segment: SG (left) compared with Levelt (1989:188) (right)

As already indicated above, content words are represented as lexical segments where the arc is labeled *head*; function words have different grammatical relations (see below). Table 4.2 lists the types of lexical segments that are distinguished for content words:

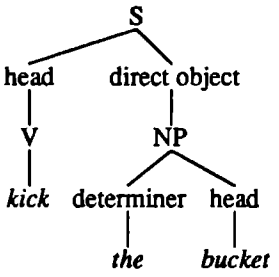
Table 4.2
Types of lexical segments for content words

nominal	NP-head-NOUN
verbal	S-head-VERB
prepositional	PP-head-PREPOSITION
adjectival	ADJP-head-ADJECTIVE
adverbial	ADVP-head-ADVERB

4.3.2 A phrasal lexicon

The lexicon as a whole consists of a set of lexical entries which are mostly single lexical segments. Because lexical segments always link a word to a phrase, the lexicon is thus essentially a *phrasal* lexicon. In such a lexicon it is straightforward to include multi-word phrases in the form of ready-made furcations of segments. In this way, idioms and frequently used syntactic fragments can readily be stored. Idioms and other multi-word phrases are thus more or less fully specified syntactic structures[5] comparable to the trees proposed in TAG by Abeillé & Schabes (1989). Example (6) shows a multi-word idiom in the lexicon for the expression *to kick the bucket*.

(6)



An SG further contains a set of lexical segments for functorization (functor segments) and a set of non-lexical segments for intercalation (intercalating segments). Since these segments have in principle the same form as the segments in the lexicon, there is no true distinction between the 'grammar' and the 'lexicon'. All language-specific knowledge is contained in a broad lexicon of segments. Hence an SG is a *lexicalized* grammar (cf. Schabes, Abeillé & Joshi, 1988).

4.3.3 Lexical rules

SG is also a *lexicalist* grammar in the sense that the lexicon is an autonomous component where lexical entries are generated by lexical rules. These processes include, e.g., passivization (computing a passive lexical entry from an active one), nominalization (computing a nominal lexical entry from a verbal one) and other derivational processes. All lexically governed, bounded, structure-preserving processes are indeed treated lexically (cf. Bresnan, 1978, 1982). Thus SG disavows structure-changing transformations in favor of having just one level of f-structure. Lexical rules operate by deriving new lexical entries from other, existing entries. These rules are associated with lemmas. E.g., the rule creating a passive counterpart of an active sentence lemma is in essence quite similar to Bresnan's (1982) passive rule:

---

[5] For ease of storage, the current implementation stores multi-segment expressions as a collection of individual segments which are not composed until they are activated.

(7)

SUBJ	→	OBL by / 0
OBJ	→	SUBJ
0	→	Passive Participle

The lexical selection process will activate existing lexical entries, or, if needed, generate new ones by means of lexical rules such as passivization or nominalization. E.g., when a verbal lemma is to unify with a NP node in the f-structure, nominalization is attempted. In order to save computational effort, this is performed in a *lazy* manner: lexical rules create new entries only when needed. Details on the implementation of the passive rule are given in Section 8.3.5.

4.4 The generation of c-structures

F-structures, as constructed by the process explained above, express ID relations but do not express any left-to-right order of constituents. By way of example, f-structure (2e) is assigned to the Dutch sentences (2a-d). F-structures are complete if the head and other obligatory segments (according to the valence information in the phrasal nodes) have been incorporated and when the functorization process has taken place. The assignment of left-to-right positions to constituents is modeled as the piecemeal derivation of a different kind of structure—a c-structure.

Somewhat like ID/LP format for PS rules, SG handles ID and LP separately. However, there are two crucial differences. First, whereas a PS-based system specifies a *relative* ordering of sister nodes, SG assigns a position to a constituent independently of its sisters; therefore, a system of *absolute* positions is used. Second, the assignment of LP in SG may be attended by a revision of ID relations. Consequently, the ID relations in the f-structure and those in the c-structure for a sentence may not be isomorphic. E.g., c-structure (2f) is assigned to (2a). For clarity, the internal details of the relative clause are left out.

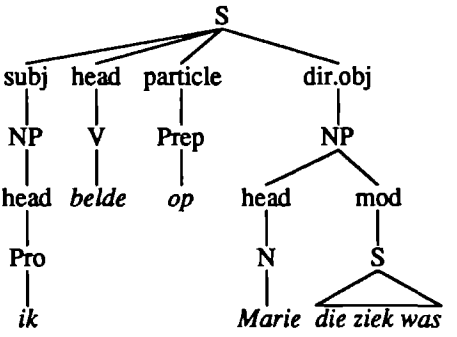
(2) a. Ik belde Marie op, die ziek was. (I called up Mary, who was ill)

b. Ik belde Marie, die ziek was, op. (id.)

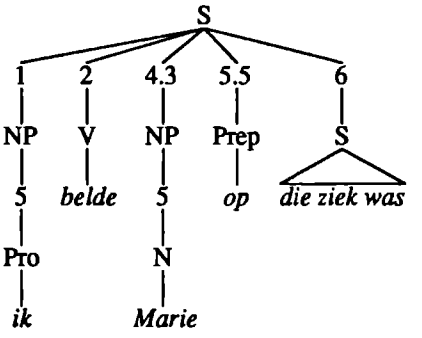
c. Marie, die ziek was, belde ik op. (id.)

d. Marie belde ik op, die ziek was. (id.)

e.



f.



4.4.1 Destinations

The procedure which assigns left-to-right positions works in a bottom-up fashion: the foot node of a segment is attached in the c-structure directly under its *destination*, which is normally the root of the segment. The destination of a constituent is determined by its mother in the f-structure, i.e., the root of the segment where the constituent is the foot. Normally, the address which the mother assigns as destination of its dependents is itself, i.e., ID relations in the c-structure are by default the same as those in the corresponding f-structure. Figure 4.8 is a schematic representation of the destination procedure.

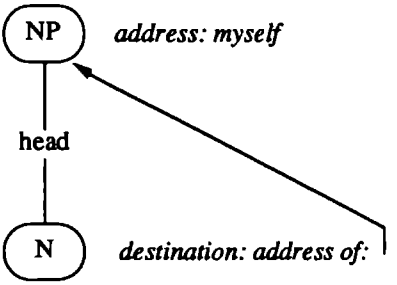


Figure 4.8
Finding the destination of a node via the address of its mother

Such indirect determination of the destination may seem complicated, but it guarantees that the *root* node of a segment in f-structure exerts control over the ID relation of the *foot* node. This will prove useful in the treatment of constructions where nodes go to higher-level destinations, e.g. the extraposed relative clause in (8a,d). We will return to this in Chapter 5.

4.4.2 Holders

Since f-structures are constructed in a piecemeal fashion, it is natural to assign word order incrementally as well. As soon as a node has been lexicalized and attached to its mother in the f-structure, IPF attempts to assign it a left-to-right position in the corresponding c-structure. Because not all constituents are available at the same time, it is difficult to encode word order relative to other constituents. Therefore, SG prefers an *absolute* order of constituents. For this purpose, a *holder* is associated with each phrase. A holder is a vector of numbered slots that can be 'filled' by constituents. Figure 4.9 shows some holders associated with c-structure (8f).

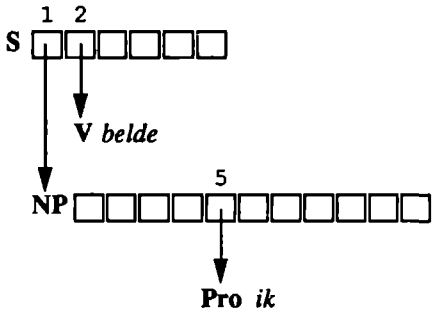


Figure 4.9
Diagram showing some holders for (8f); the first and second positions of the S have just been occupied

The foot node of each segment in the grammar has a feature '*positions*' which lists all possible positions that the node can occupy in its destination. These possibilities can be seen as language-specific constraints on word order variation. Constituents will try to occupy the first available slot. E.g., the subject NP in Dutch may occur either in sentence-initial or third position (cf. examples (8a) vs. (8c)). In the grammar for Dutch this is specified so that the foot of an S-subject-NP segment may go to holder slots 1 or 3. When a subject NP is assigned a position in the holder of its destination, it will first attempt to occupy position 1. Suppose that position 1 has already been occupied by another constituent, due to earlier conceptual input, it will attempt to go to position 3. Such a situation, which is schematically represented in Figure 4.10, may give rise to word order variations like (8c,d), where *ik* takes third position rather than first.

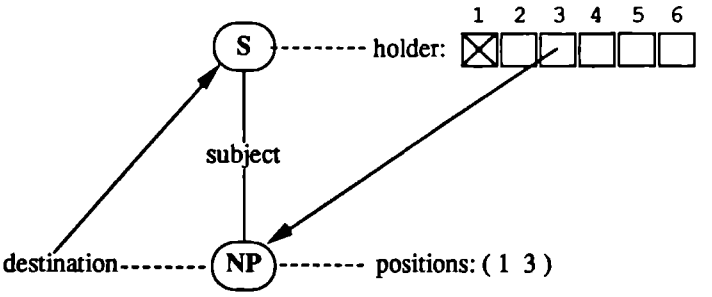


Figure 4.10
Destination and linearization processes: assign NP to the third slot in the holder of its destination because the first slot is already taken

If the utterance has proceeded beyond the point where a constituent can be added, a syntactic dead-end occurs and a self-correction or restart may be necessary. Alternatively, the constituent may end up somewhere else. E.g., the final slot in the S holder is a 'dump' for late material (*right dislocation*). The relative clause in (8a,d) is an instance of such a construction (see also Section 5.2).

Although human speakers do not always make perfect sentences, and sometimes produce clear ordering errors, it seems generally possible, at least in languages like English and Dutch, to determine the order of single fragments one after the other during incremental production. In fact, it seems that *incremental production is only possible if for the assignment of left-to-right position to a constituent, the simultaneous presence of other constituents of the phrase is not required.* If this empirical claim is true, the necessary knowledge to determine word order can be encoded locally at the level of single segments, as is done in SG.

The number of holder slots in a Dutch clause is substantial. In order to keep an overview, positions *within* positions are sometimes used. The Dutch sentence can be divided into six main parts, each having its own internal ordering. Decimal notation is used to represent such slots, e.g., *3.2* is the second slot in the third main slot. Some holder slots can be occupied by a single constituent only; others may be occupied by an

unspecified number of constituents, e.g., an indefinitely long list of APs in front of a noun.

Table 4.3 shows an overview of word order positions currently used in the Dutch SG used by IPF. There is no pretension to be complete. The holder slots of the S have been inspired by De Schutter (1976) who distinguishes five main slots, each slot subdivided into other ones. The positional frame of the NP has been adapted for Dutch from the English grammar by Dekeyser, Devriendt, Tops and Geukens (1979).

Table 4.3
Overview of left-to-right positions in main categories

S

1 thematic or focused constituent, interrogative pronoun in Wh-question or empty in Yes/No question (main clause), preposition (subclause)
2 finite verb (main clause), subordinating conjunction (subclause)
3 1 pronominal subject, situating *er*
2 ADVPs, PPs (modifier, indirect object, passive by-phrase)
3 non-pronominal subject
4 direct object pronouns, partitive *er*
5 indirect object pronouns, reflexive, *er* separated from a preposition (*er ... voor*)
6 demonstrative pronouns as objects
4 1 indirect object (no PP)
2 direct object (definite, no pronoun), ADVPs
3 conjunctive ADVP (*echter, immers*), modal ADVP (*misschien, helaas*)
4 ADVPs, PPs (modifier, indirect object, passive by-phrase), illocutive particles (*maar, toch, nog*), propositional particles (*slechts, maar, ook, alleen, niet*)
5 direct object (indefinite, no pronoun)
5 1 preposition as separated from a pronominal adverb (*er ... voor*)
2 ADVPs, PPs (modifier, indirect object, passive by-phrase)
3 prepositional object, including inherent modifier (*naar huis (gaan), in vier stukken (snijden)*), and obligatory phrases in idioms (*op de vuist (gaan)*)
4 subject complement, object complement, first part of a compound verb
5 clause-final verb cluster
6 list of constituents, finite subclause, ADVP (modifier), PP (modifier, indirect object, passive by-phrase), dislocated constituent

NP

1 initiator (*al (de), alle, zo (een)*, interrogative pronoun, (*welke, welk (een)*)
2 determiner: deictic (demonstrative pronoun), article, quantifier (*sommige*), possessive pronoun, possessive adnominal, specifying genitive
3 quantifier (*vele*), limiter (*voornaamste, dergelijke*), cardinal or ordinal number (*drie, derde*)
4 adjective
1 evaluative (or subjective, *mooi*)

2 descriptive (or objective)
1 relative
1 size (*klein*)
2 age (*oud*)
2 absolute
1 color, shape (*rood, langwerpig*)
2 material (*houten*)
3 participles, te+infinitive (*beschilderd, te beschilderen*)
4 provenance modifier (*Spaans*)
3 defining modifier (*gotisch, Renaissance*)
5 head (*kistje*)
6 postposed adnominal noun (*juwelen*)
7 PP, ADVP
8 infinitival adjectival clause with *te, om ... te*, ADJP (especially participles)
9 restrictive relative clause, finite adjectival clause with *dat, of*
10 apposition (NP or S)
11 non-restrictive relative clause

AP
1 modifier
2 head

PP
1 modifier (*dwars*)
2 head (*door*)
3 object (*de muur*)
4 postposition (*heen*)

4.4.3 Topicalization, accessibility and word order variation

Variations in word order, e.g. (9), do not have one unique origin in IPF, but are determined by an interplay of several timing factors in the generation process.

(9) a. Jan speelde in Amsterdam vorige week.
(John played in Amsterdam last week)

b. Jan speelde vorige week in Amsterdam.

c. Vorige week speelde Jan in Amsterdam.

d. In Amsterdam speelde Jan vorige week.

The first factor is *conceptual accessibility*, or the time at which which preverbal messages are generated by the Conceptualizer and are passed to the Formulator. The sooner a conceptual fragment is entered into the Grammatical Encoder, the sooner it

can be syntactically processed. Due to the principle that the earliest (leftmost) possible position is assigned to a constituent, those constituents which are processed earlier tend to come earlier in the sentence. The notion of conceptual accessibility is assumed to correspond closely with topicalization (cf. Bock & Warren, 1985). The more a concept is topic in the discourse, the more it is conceptually accessible. Topic fronting in languages like Dutch and English can therefore be seen as an emerging property of the process rather than as an explicit rule.

A second, similar factor is *lexical* accessibility: the sooner a lexical item (lemma) is found, the sooner the Grammatical Encoder can incorporate it in the syntactic structure.

A third factor is the *complexity* of the concept to be realized. Assuming that long and complex constituents consume more processing time, their grammatical encoding is likely to finish later than those of other, simpler constituents which have started at the same time or even sooner. Again, this factor is essentially one involving duration: the longer a process takes, the more likely it is that other processes, running in parallel and completing sooner, will have taken up earlier positions. Thus, the tendency of long and complex constituents to occur at the end of the sentence is not a rule of linguistic structure, but an emerging property of the generation process as it is proposed here.

Experience with the implementation of IPF shows that these timing factors can be well simulated in a program based on a pseudo-parallel computer system. Alternative orderings of the same conceptual input stimuli into the Formulator can indeed result in word order variations on the surface level. E.g., the utterances (10) can be generated by IPF under circumstances which differed solely by the fact that in (10a) the concept underlying *Otto* is accessible sooner than in (10b), while in (10b) the concept underlying *appel* (apple) is given sooner. If the concepts are input at roughly the same time, the chances of (10a) or (10b) being generated are about equal.

(10) a. Otto ... heeft een appel gegeten.
(Otto has eaten an apple)

b. Een appel ... heeft Otto gegeten.
(id.)

Similarly, in SVO languages like English, passivization can be triggered by conceptual accessibility. If a constituent which would normally be object in an active sentence is fronted, it becomes a likely candidate for the subject function; hence a passive lemma is appropriate.

Summing up, if there is indeed a strong link between fronting, topicalization and conceptual accessibility (Bock, 1987; Bock & Warren, 1985; Levelt, 1989) then topicalization by fronting (and also by means of some other mechanisms like passivization) is an important emerging property of the generation strategy. This is not to say that accessibility is the *only* factor determining the order in which conceptual fragments are passed to the Formulator. It seems that certain rhetorical effects unrelated to accessibility may affect word order as well. Kolk's (1987) adaptation theory suggests that the conceptual input to the Formulator can be subject to manipulation which is learned by the speaker. Thus, it is very well possible that certain

factors determining word order are coded in terms of an order imposed on conceptual elements as they enter the Formulator.

Although experiments with IPF show that conceptual accessibility can indeed account for variations in word order, the timing of the parallel processes in the Grammatical Encoder is not very sophisticated, since the current implementation gives all parallel processes the same priorities; they are therefore capable to do roughly the same proportion of processing in the same period of time. In a multiprocessing environment it is often possible to allot more computational resources to one process than to another. This would be another way to affect word order by means of timing in the generation process.

4.5 An IPF example

In order to make the foregoing mechanisms more concrete, we now present a small example of an IPF simulation. IPF is implemented in CommonORBIT on a Symbolics Lisp Machine. A program frame supporting multiple windows is used to display intermediate results of the generation of a sentence (see Figures 4.11 and 4.12). F-structures are displayed in the upper left window while c-structures are displayed in the lower left window. These windows scroll up as new information is displayed at the bottom. By watching successive structures appear in the windows, one gets an impression of the progress of the generation process over time. In the lower right window, a summary of actions is presented.

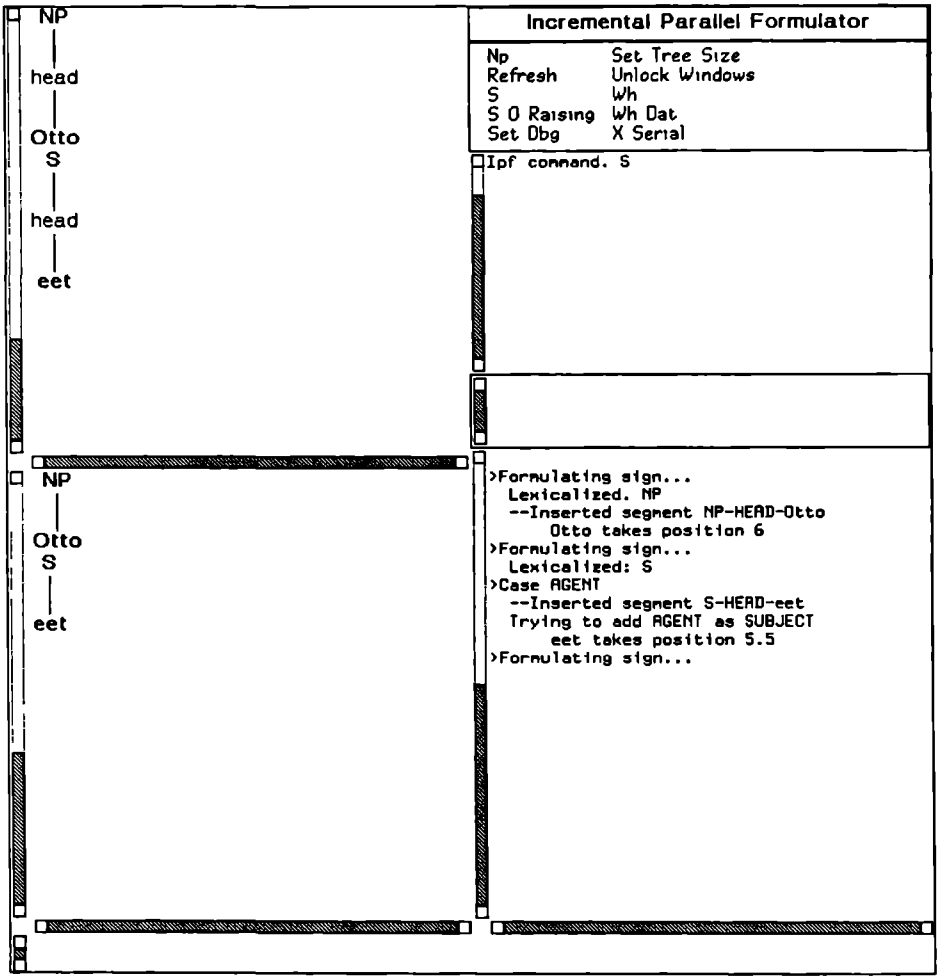
The example will involve the generation of (10a) and (10b) and hopes to make clear that the timing of input to the Formulator can influence the assignment of left-to-right order of constituents in the c-structure. The input of the first simulation consists of the following LISP expression:

```
(DEFUN S-TEST ()
  "Simulation of: Otto heeft een appel gegeten."
  (LET ((SIGN1 (A SIGN
                  (SEMANTIC-REFERENT (THE-OBJECT 'OTTO))))
        (SIGN2 (A SIGN
                  (SEMANTIC-REFERENT (AN EAT))))
        (SIGN3 (A SIGN
                  (SEMANTIC-REFERENT (AN APPLE)))))
    ;; first define features; these could be computed
    (DEFINE-FEATURES SIGN1
      '(DEFINITE + PLURAL -))
    (DEFINE-FEATURES SIGN2
      '(INTERROGATIVE - PERFECT + FUTURE - PAST - FINITE +))
    (DEFINE-FEATURES SIGN3
      '(DEFINITE - PLURAL -))
    (FORMULATE SIGN1)
    ;; Otto...
    (SLEEP 5)
    (FORMULATE SIGN2)
    ;; heeft gegeten ...
    (SLEEP 5)
    (DEFINE-CASE SIGN1 'AGENT :IN SIGN2)
    ;; Otto heeft gegeten ... = upward expansion
    (SLEEP 5)
    (FORMULATE SIGN3)
    ;; een appel
    (SLEEP 5)
    (DEFINE-CASE SIGN3 'THEME :IN SIGN2)
    ;; Otto heeft een appel gegeten ... = downward expansion
    (LIST SIGN1 SIGN2 SIGN3)))
```

This input first creates three empty signs and then sends a number of commands to the Formulator. The DEFINE-FEATURES commands simply assign the features to the linguistic sign. More substantial commands are three involving FORMULATE and two involving DEFINE-CASE. Each command to the Formulator immediately spawns an independent grammatical encoding process which runs in parallel to everything else going on in the system. However, the commands in this simulation are spaced in the time dimension by means of SLEEP commands which each cause the system to be dormant for approximately 5 seconds of real time. Due to these time delays, the generator will profit more from its incremental mode of generation.

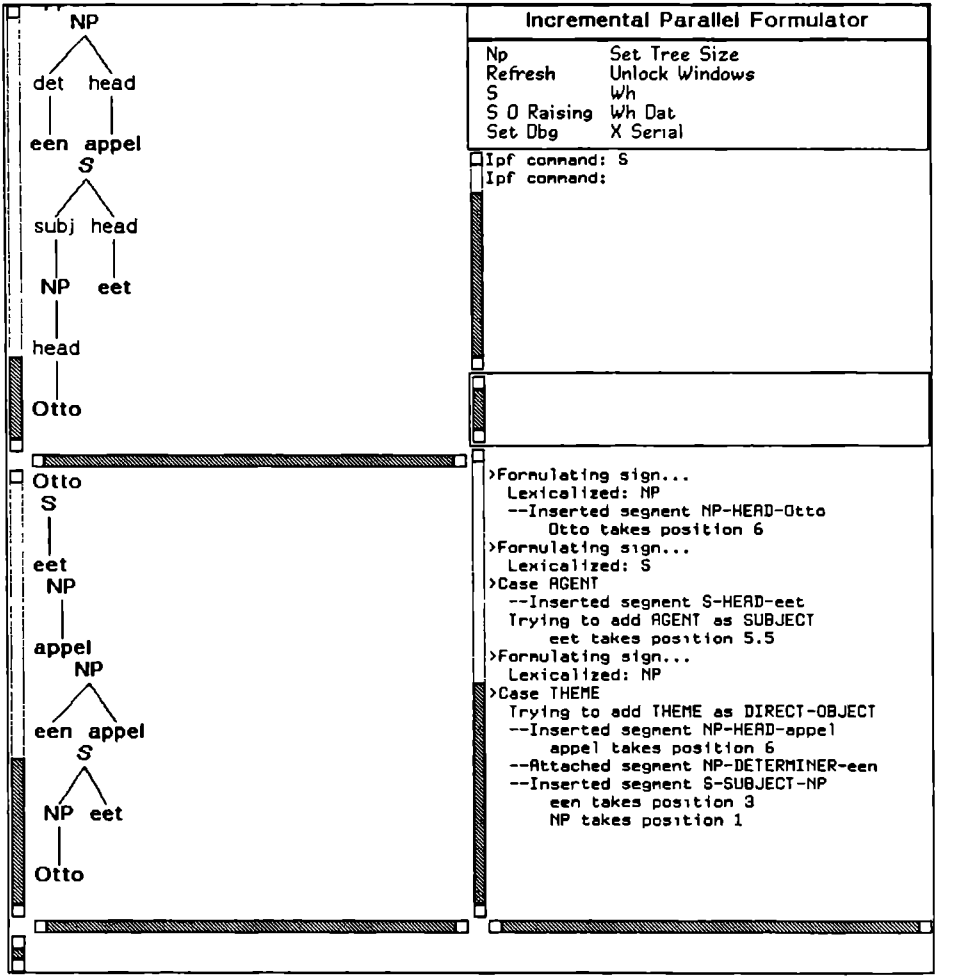
Figure 4.11 presents three snapshots of the generation process. Due to the large time gap between the inputs, the sign for the concept *Otto* 'stays ahead' of the sign for *apple* and occupies a position earlier in the sentence. This results in a c-structure corresponding to sentence (10a). The left-to-right order of the NPs reflects the order of the conceptual fragments in the input.

a.[6]



[6] The positional encoding used in this output differs slightly from that presented in Table 4.3. In particular, the head N takes position 6 in the NP rather than 5

b.


NP
det head
een appel
S
subj head
NP eet
head
Otto
Incremental Parallel Formulator
Np Set Tree Size
Refresh Unlock Windows
S Wh
S O Raising Wh Dat
Set Dbg X Serial
Ipf command: S
Ipf command:
Otto
S
eet
NP
appel
NP
een appel
S
NP eet
Otto
>Formulating sign...
Lexicalized: NP
--Inserted segment NP-HEAD-Otto
Otto takes position 6
>Formulating sign...
Lexicalized: S
>Case AGENT
--Inserted segment S-HEAD-eet
Trying to add AGENT as SUBJECT
eet takes position 5.5
>Formulating sign...
Lexicalized: NP
>Case THEME
Trying to add THEME as DIRECT-OBJECT
--Inserted segment NP-HEAD-appel
appel takes position 6
--Attached segment NP-DETERMINER-een
--Inserted segment S-SUBJECT-NP
een takes position 3
NP takes position 1

c.

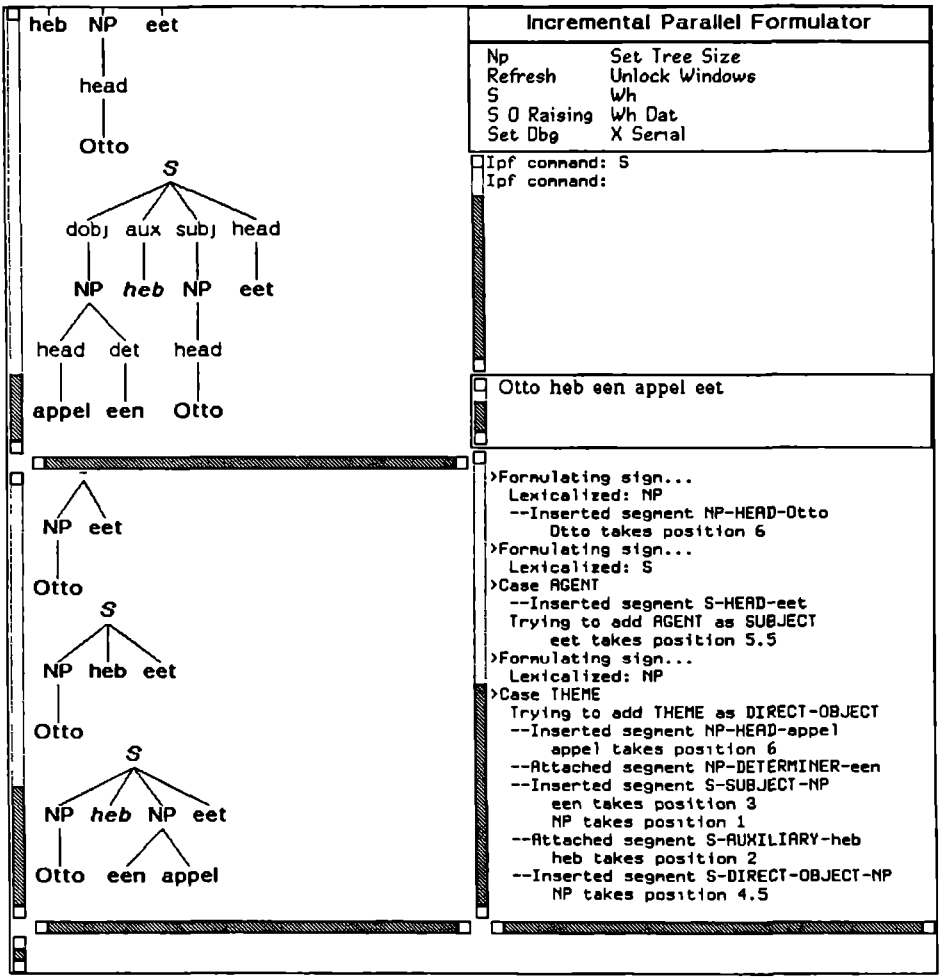
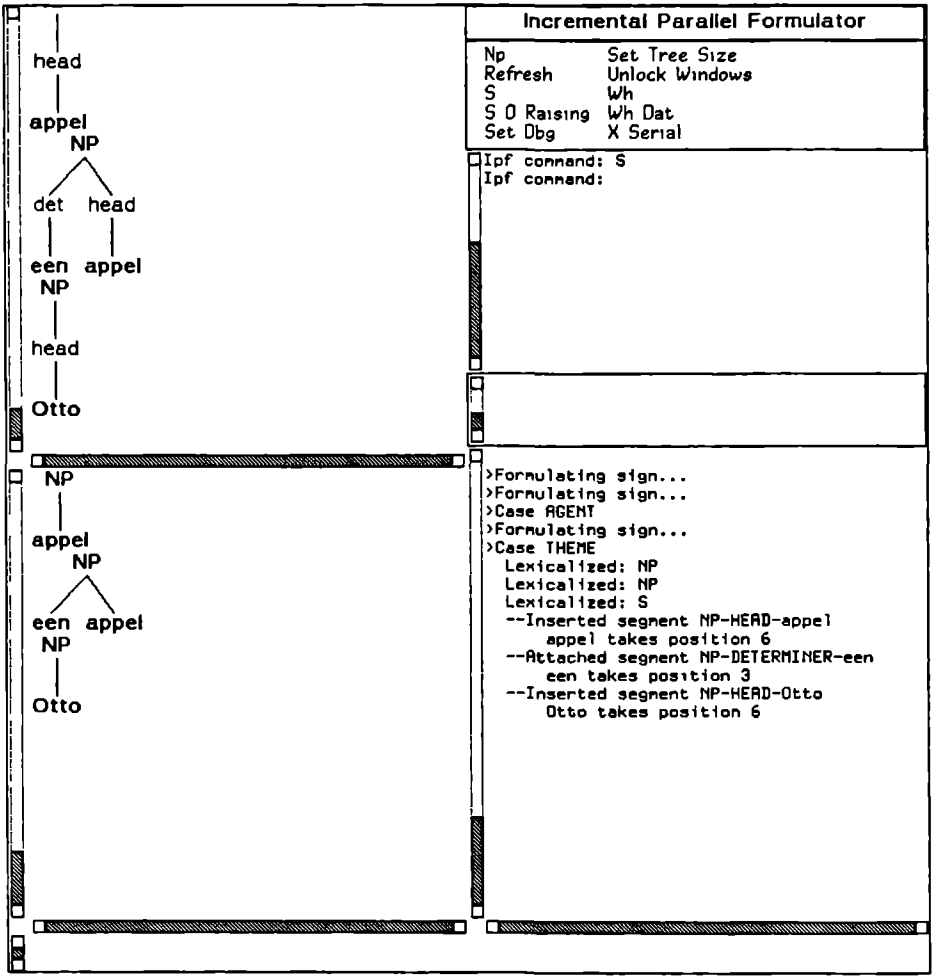


Figure 4.11
Three snapshots of the generation of example (10a)

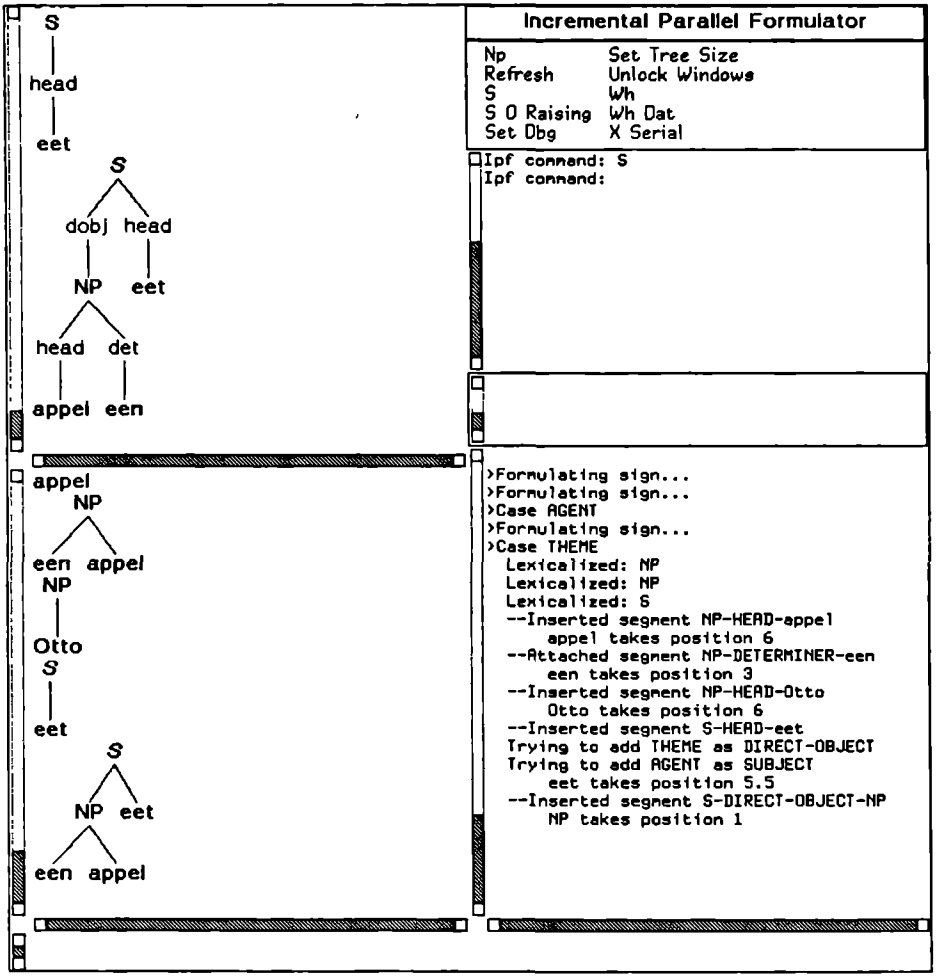
In a different simulation run, the SLEEP commands in the input were removed. Consequently, the computation of parallel encoding processes overlap more, and the constituents of the sentence therefore have a higher chance of occupying alternative left-to-right positions in the c-structure. In other words, left-to-right order in the sentence will correspond less to the order of the conceptual fragments in the input and will therefore be less predictable. This is illustrated in Figure 4.12. The trace displayed in the lower right window shows how closely spaced the conceptual messages are (their traces are each preceded by the character '>'; compare Figure 4.11). The sign for *Otto*

happens to be 'overtaken' by that for *apple* and ends up in holder slot 3 of the S. This results in a c-structure corresponding to sentence (10b).

a.



b.


S
head
eet
S
dobj head
NP eet
head det
appel een
Incremental Parallel Formulator
Np Set Tree Size
Refresh Unlock Windows
S Wh
S O Raising Wh Dat
Set Dbg X Serial
Ipf command: S
Ipf command:
appel
NP
een appel
NP
Otto
S
eet
S
NP eet
een appel
>Formulating sign...
>Formulating sign...
>Case AGENT
>Formulating sign...
>Case THEME
Lexicalized: NP
Lexicalized: NP
Lexicalized: S
--Inserted segment NP-HEAD-appel
appel takes position 6
--Attached segment NP-DETERMINER-een
een takes position 3
--Inserted segment NP-HEAD-Otto
Otto takes position 6
--Inserted segment S-HEAD-eet
Trying to add THEME as DIRECT-OBJECT
Trying to add AGENT as SUBJECT
eet takes position 5.5
--Inserted segment S-DIRECT-OBJECT-NP
NP takes position 1

c.

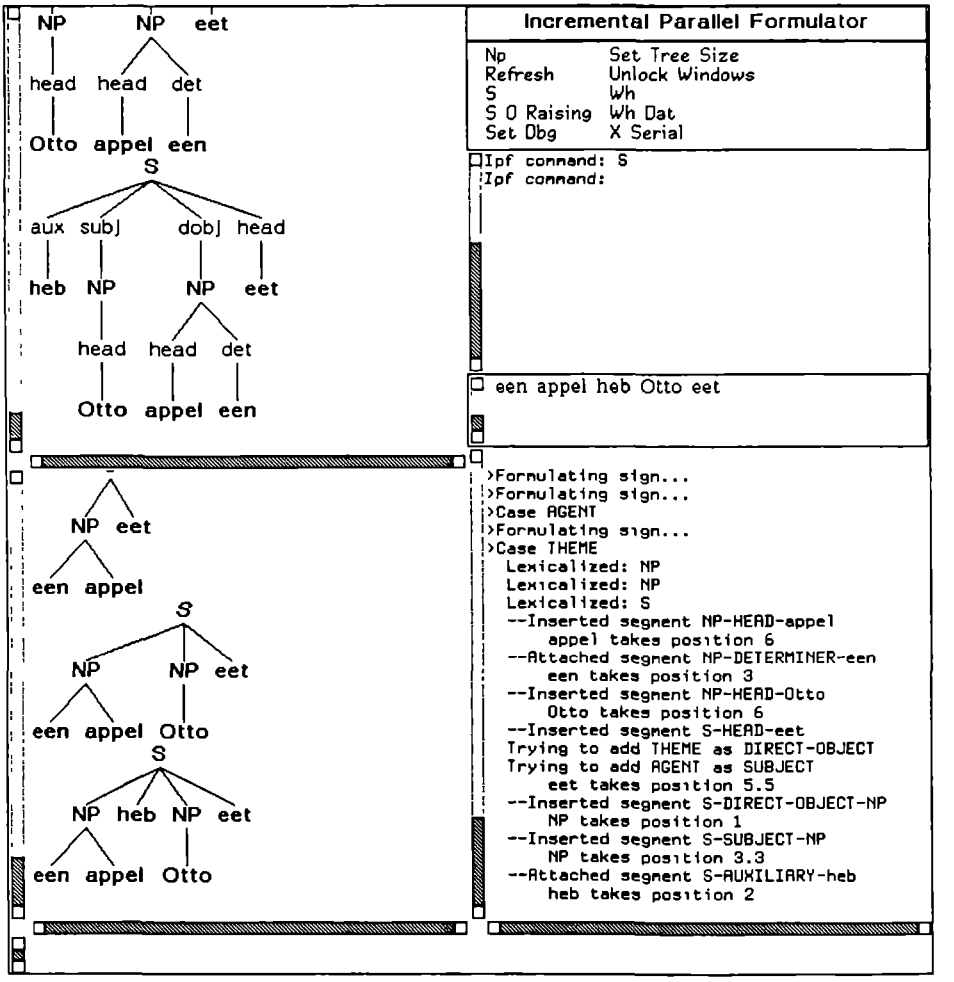


Figure 4.12
Three snapshots of the generation of example (10b)

4.6 Concluding remarks

IPF is a computer model for incremental grammatical encoding which exploits the representation of grammatical and lexical knowledge in terms of syntactic segments. It allows the piecemeal input of conceptual material and can construct functional structures not only from the top down, but also from the bottom up. Surface structures (c-structures) are derived in an equally piecemeal fashion.

Clearly, a parallel formulator has an advantage compared to one which operates in a sequential mode. Newly incoming fragments which take little processing time can be uttered before older but more difficult fragments are ready. This suggests that the time when a fragment can be uttered depends not only on the moment when it has entered the formulator, but also on how much processing time the formulator spends on it. Extraposition of long and complicated phrases at the end of the sentence, which is often found in spontaneous speech, is indeed predicted by this model.

Although conceptual changes other than addition of new concepts are currently not implemented in IPF, they are in principle possible. Conceptual replacement or deletion of concepts, semantic roles or features could be modeled either by undoing previous unifications or by the construction of a new sentence structure. The latter seems to be preferred because undoing previous unifications involves a great deal of bookkeeping of past states of the unification space. When a conceptual change invalidates a piece of structure (which may be partially uttered), the structure could be abandoned and reconstructed to model a restart.

A point of criticism which could be made with respect to the semantic/syntactic interface in IPF is the implicit assumption that the underlying conceptual structure is nearly isomorphic with the f-structure. Although there are provisions for multi-word phrases to cover a single concept, the reverse is not possible yet: a complex conceptual structure cannot at present be rendered as a single lexical entry. This has serious implications for incremental generation, because lexicalization might have to proceed in an incremental fashion as well. When part of the lexical content of a word is input to the Formulator, lexicalization may have to wait until more content is available to perform the correct selection of a lexical entry. An investigation of this possibility remains outside the scope of this work.

A general question which so far has remained unanswered is how the output of the Grammatical Encoder is passed on to the next processing module, the Phonological Encoder, and how the output of the latter is passed on to the Articulator. All information to compute phonological word forms is contained in the c-structures:

1 Each word in the c-structure contains a stem and a number of features to guide inflectional and phonological processes;
2 The left-to-right relations in the holders allow left-to-right phonological processing and articulation;
3 The ID relations in the c-structure are used for the assignment of prosodic structure (rhythm and intonation contour).

In principle, left-to-right processing of the c-structure by the Phonological Encoder can be achieved by traversing this structure depth-first. However, since c-structures are built incrementally, the Phonological Encoder can take advantage of this by proceeding in an incremental fashion as well. The question is, to what extent incremental phonological encoding is possible. Can the Phonological Encoder process every little piece at a time and pass it on to the Articulator as soon as possible, or should it wait until a reasonable sentence structure has emerged? We believe that there is no absolute answer to this question, but that both are possible to some extent. People sometimes produce very spontaneous speech and sometimes very careful and deliberate speech. It

seems that a main determinant of these modes of speech is how the Phonological Encoder interfaces to the output of the Grammatical Encoder. A solution is suggested in Chapter 10.

It could be assumed that the Phonological Encoder normally applies certain heuristics to plan its processing. At any rate, there should be a mechanism to prevent 'skipping over' the position of required constituents, e.g., the finite verb of a finite clause. One solution which has been adopted in the current implementation consists of marking the required slots in a c-structure with a special symbol to signal that this position *must* contain a constituent. When the Phonological Encoder finds such a marker, it waits until the position is filled by a constituent which overwrites the marker.

A remaining question is, when a new sentence is to be started. In an incremental mode of sentence generation, the answer to this question is not straightforward, for a sentence which is in principle complete may often be extended by adding another modifier or even a case relation, e.g.:

(11) a. He came to my house ... at about ten o'clock ... because he needed money.

b. By ten I decided to leave ... the town.

A new sentence seems to be triggered by at least two kinds of conceptual addition:

1 When a new concept is not related to anything in the conceptual structure underlying the current sentence.

2 When a new concept is related to the current utterance, but does not fit into the syntactic structure, because the utterance has proceeded beyond the point where the new constituent can be added (as discussed above). In that case, either the sentence is already complete, and a new one is started, or the current structure is abandoned and a restart is attempted.

A final remark concerns the encoding of word order which has been proposed in this chapter. It seems unlikely that the proposed format can easily be acquired by language learners, for a small change may require a major recoding of absolute positions involving many segments. Also, it seems that many constraints on word order (e.g. sentence modifiers) are so subtle that they are difficult to express as hard rules (cf. Florijn, in preparation). As an alternative, the proposed absolute positions might be recoded, e.g., as vectors of binary features, so that they can be learned by means of a connectionist network. A connectionist approach could account not only for learning, but also for slight errors and variations in word order.

5 Discontinuous constituency in Segment Grammar[1]

In order to illustrate some of the grammatical coverage of SG, and, in particular, to demonstrate how the machinery developed in the previous chapter is put to work, an account will be given of some special syntactic constructions in Dutch. Given the diversity of syntactic constructions in natural languages, it is beyond the scope of this study to work out a substantial SG for a language, so I will restrict myself to a discussion of discontinuous constituency in Dutch.

In Chapter 4, it was already indicated how the assignment of left-to-right positions may be accompanied by changes in the immediate dominance relations. Thus, a c-structure need not be isomorphic to the corresponding f-structure. SG accounts for various kinds of discontinuous constituency—including right dislocation, S-O raising, and unbounded dependencies—by assigning some constituents different ID relations in the c-structure than those in the f-structure. Constructions which are treated in this way include clause union, right dislocation, and unbounded dependencies. Separable parts of words such as separable verbs and compound prepositions are not viewed as true discontinuities but as lexical entries consisting of separate syntactic segments.

5.1 Introduction

Even languages with a relatively fixed word order allow constituents of a phrase to be non-adjacent. In (1)-(5), five important kinds of discontinuities in Dutch are summed up (cf. Bunt, 1988).

The examples in (1) contain broken-up constituents which in part have been dislocated to the right, across another constituent.

(1) a. Ik heb een auto gekocht met zes deuren.
(I have bought a car with six doors)

b. Een van zijn vrienden kwam, die in Brussel woont.
(One of his friends came, who lives in Brussels)

c. Een betere film dan ik verwachtte draaide in Calypso.
(A better film than I expected was shown at Calypso)

d. Een betere film draaide in Calypso dan ik verwachtte.
(id.)

[1] An earlier version of this chapter has been published as (De Smedt & Kempen, 1990).

Example (1a) shows a discontinuous NP with an extraposed PP. Extraposition is optional and tends to occur more often in spontaneous speech. Example (1b) shows a similar construction with a right dislocated relative clause rather than a PP. Again, extraposition is optional, but tends to be more acceptable as the relative clause is longer and the rest of the sentence (*kwam*) is shorter. In (1c), it is an adjectival phrase (ADJP) which is discontinuous; this right dislocation is obligatory and can extend not only to the NP level (1c) but also to the S level (1d).

A second kind of discontinuity consists of compound words which are separable, e.g the verb *opbellen* in (2a) and the compound preposition *doorheen* (through) in (2b).

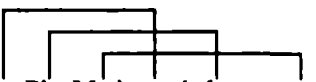
(2) a. Bel me morgen op.
(Call me up tomorrow)

b. Het vliegtuig gaat nu door de geluidsbarrière heen.
(The airplane now breaks the sound barrier)

In SG, these cases are not treated as discontinuities at all, but as lexical entries which consist of several segments, so that the 'separated' elements are *already* separate in f-structure. They may be assigned left-to-right positions in such a way that other constituents may intervene. The French negative *ne ... pas* is also an instance of this kind of lexical entry.

Examples of *clause union* are given in (3). In (3a) the constituents of the infinitival clause (underlined) are not kept together as a whole but are assigned positions in the main clause. Thus objects are grouped irrespective of the clause where they functionally belong, and likewise for non-finite verbs. Clause union may result in crossed dependencies in Dutch, as shown in (3b).

(3) a. Ik heb Otto een appel zien eten.
(I have seen Otto eat an apple)



b. Ik dacht dat Jan Piet Marie zag helpen zwemmen
(I thought that Jan saw Piet help Mary swim)

A fourth kind of discontinuities involves unbounded dependencies, e.g., Wh-extraction in (4a) and fronting of a focused element in (4b) from subordinate clauses.

(4) a. Wie dacht je dat ik opbelde?
(Who did you think I called?)

b. Dat blonde meisje dacht ik dat hij opbelde.
(That blond girl I thought he called)

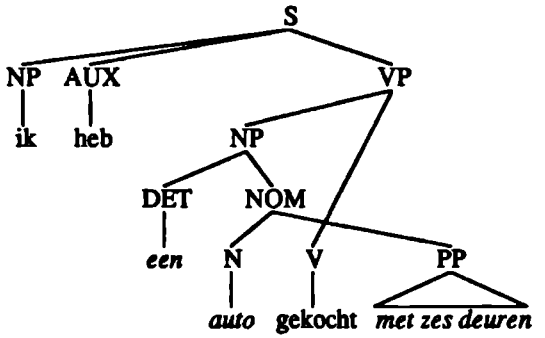
A fifth kind of discontinuity in Dutch contains *er* (there) and similar 'R-words'. When the object in a PP does not have a person as its antecedent, it is pronominalized by means of the special pronoun *er* (often also called a pronominal adverb), which is

placed before the preposition. This combination of *er* and a preposition may be interrupted by some other constituents, as in (5).

(5) De vloeistof gaat er nu in.
("The liquid goes there now in", The liquid now goes in it)

Since these various kinds of discontinuous constructions present problems for PS-based grammars, it has been proposed to amend the definition of an ordinary PS tree to accommodate discontinuities. For sentence (1a) this could result in the modified tree structure (6).

(6)

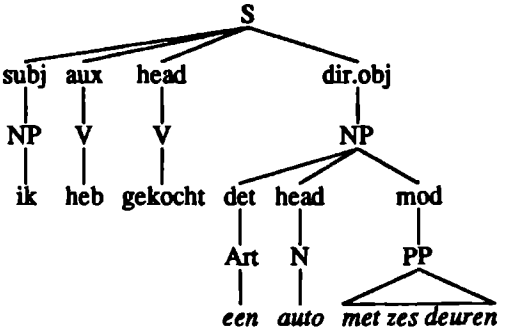


In order to generate such structures within a context-free framework, Bunt (1988) proposes Discontinuous Phrase Structure Grammar (DPSG) which introduces and formalizes the notion of adjacency into PS grammars. DPSG is motivated by the claim that other generation algorithms for a language with discontinuities would first have to generate a continuous ordered tree representation and then identify and apply the transformations which produce the correct word order for discontinuities. However, this need not be the case for a grammar which distinguishes between an *unordered* functional structure (f-structure) and an ordered surface structure (c-structure). In SG, the correct word order is produced directly. In the next sections it is shown how various discontinuities are handled by changes in ID relations at the time when left-to-right order is determined.

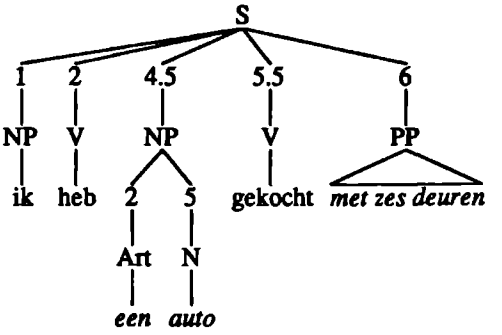
5.2 Right dislocation

At the level of f-structure, SG assigns the right dislocated PP in (3a) a functional relation to the NP (*een auto* ...) as shown in f-structure (7a). However, the PP is not part of the NP in the c-structure. Rather, it has an ID relation to the S, as shown in c-structure (7b). Since the f-structure is unordered, the computation of c-structure is not a transformation in the TGG sense, but an assignment of left-to-right order accompanied by a simple reassignment of an ID relation.

(7) a.



b.

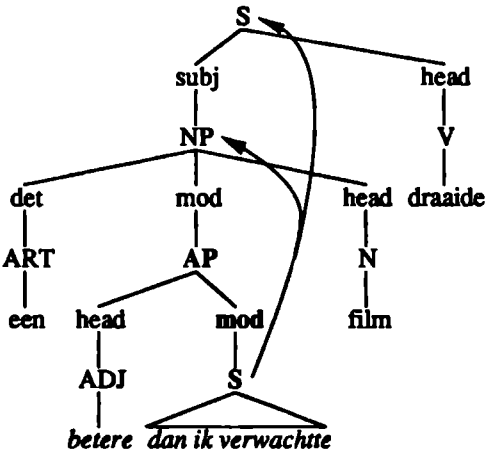


The right dislocation of a PP fits naturally into the incremental generation process. It is triggered by the fact that the extraposed PP cannot be added incrementally to the holder of the NP, because the utterance has already proceeded beyond that point. Therefore the PP is exceptionally allowed to move to the S level, which has a holder slot where some kinds of 'late' constituents can be placed. Preferably, this slot (numbered 6) contains at most one constituent; more than one constituent is not impossible though, as in (8a,b). The character of such constituents as 'afterthoughts' becomes clearer if more constituents are added.

(8) a. Ik heb een auto gekocht vorige week ... met zes deuren.
(I have bought a car last week ... with six doors)

b. Ik heb een auto gekocht met zes deuren ... vorige week.
(I have bought a car with six doors ... last week)

Right dislocation is sometimes obligatory, e.g. (1c,d). Such obligatory right dislocation must be specified in the lexicon. In SG this is achieved by a specification of possible destinations on the foot of the AP-mod-S segment which is associated with the lexical entries for *beter ... dan* and other comparatives. F-structure (9) contains a schematic indication of these possibilities.

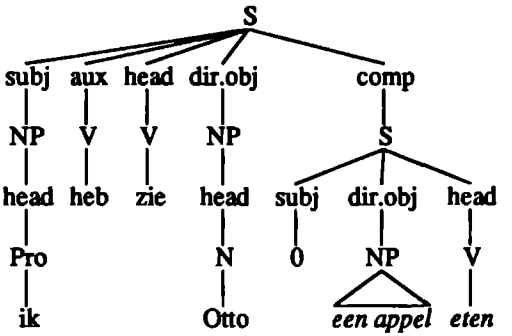


5.3 Clause union and 'raising'

This section deals with *subject-to-object raising*. This construction (henceforth *S-O raising*) is characterized by a direct object, e.g., *Otto* in (3), which simultaneously serves as the *logical subject* of an infinitival complement clause. According to SG, as well as certain other contemporary accounts, this construction does not actually involve *raising* in the transformational sense. The direct object *Otto* in (3) is always the object of the matrix clause and never subject of the embedded clause at any point during the generation process. This is compatible with the LFG analysis of such constructions, which has also been argued for independently on the basis of cross-language investigation by Horn (1985).

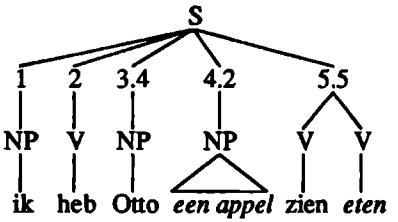
The 'raised' object must be semantically related to the matrix S as well as to the complement (*comp*) S. E.g., in sentence (3), which is assigned f-structure (10), the 'raised' direct object realizes the *theme* of the proposition expressed by the matrix S. It also holds the *agent* relation to the action expressed by the complement S. This would normally result in the addition of a subject. It is indeed a precondition that the direct object in the matrix S is coreferential with the subject in the complement S. However, the subject in the complement S is not realized because non-finite clauses never have subjects.

(10)



The discontinuity of the infinitival clause, as shown by means of underlining in (3), is accounted for by means of *clause union:* the complement S forms one surface unit with the matrix S. That is, the constituents of the embedded S are assigned positions in the holder of the matrix S. Although both infinitives collocate in one positional slot, the infinitives from deeper clauses—which are positioned later—are added at the end. The resulting c-structure is shown in (11).

(11)



Clause union is brought about by the same destination mechanism which constructs 'normal' c-structures. As mentioned in Section 4.4, the destination of a constituent is determined by its mother node in the f-structure. Normally, the mother assigns a dependent constituent a position in its own holder. However, clauses whose constituents are 'raised' do *not* function as destinations. Rather, they use *their* mother nodes as the destination address for their constituents (see Figure 5.1)[2].

[2] The implementation of this rule is shown in Section 8.3.4.

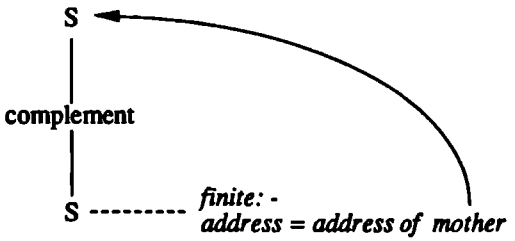


Figure 5.1
Segment for object complement clause.

Raised constituents may themselves contain raised constituents: pointers are followed step by step upward in the tree until a node is found which will function as destination address. The infinitival complement S node goes 'nowhere' in the c-structure; since all its dependent constituents are put elsewhere, there is no need to assign it a position.

S-O raising requires substantial planning ahead: if only the subject and head are planned and realized, the sentence will come out as (12), which cannot be continued as an S-O raising construction, but can—in this case—be complemented with a finite subclause (a *dat*-clause). In order for S-O raising to be successful in an incremental mode of generation, it is therefore necessary that the case relations involved in this construction are established well in advance.

(12) Ik heb gezien ... [dat hij een cake bakt].
(I have seen ... [that he bakes a cake])

The requirement of semantic coreferentiality of the direct object of the matrix clause with the subject of the embedded clause may force passivization, e.g. (13). Suppose that in an incremental mode of generation, the direct object of the matrix clause is generated first. If this constituent is coreferential with the direct object of the (active) embedded clause, then that clause cannot be realized as an active one, because coreferentiality with its subject is required. The lexicalization process may then apply lexical rules to the lemma of the embedded clause. Passivization will produce a lemma where the subject is coreferential with the direct object of the matrix clause.

(13) Jan ziet Piet ... door Marie gekust worden.
(John sees Peter ... being kissed by Mary)

5.4 Cross-serial dependencies

When multiple instances of clause union are embedded in a finite subclause in Dutch, the c-structure may exhibit so-called cross-serial dependencies. Example (14a) is taken from Bresnan, Kaplan, Peters and Zaenen (1982). The horizontal brackets indicate dependency relations between NPs and main verbs. In (14b) the vertical brackets

indicate the grouping of constituents in surface positions. The German translation equivalent (14c) shows that the ordering of the infinitives is language-specific.

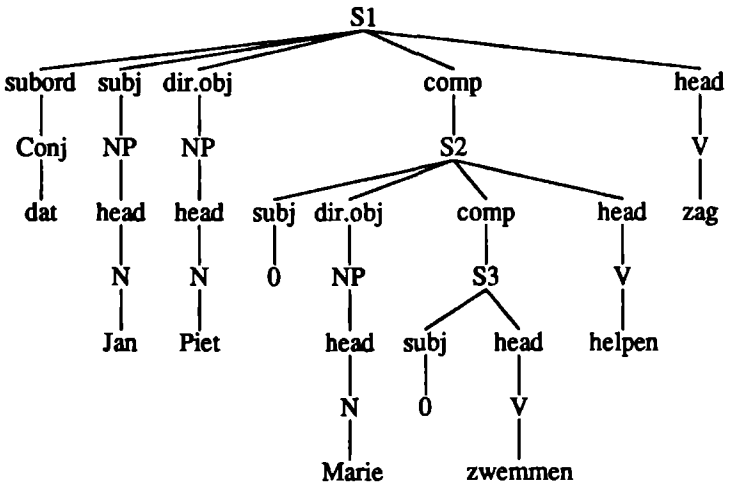
(14) a. ... dat Jan Piet Marie zag helpen zwemmen
(...that Jan saw Piet help Mary swim)

b. ... dat Jan [Piet Marie] [zag helpen zwemmen]

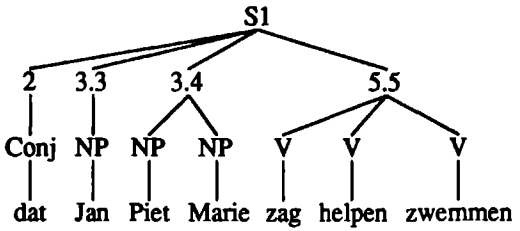
c. ... daß Jan [Piet Marie] [schwimmen helfen sah]

As shown in f-structure (15a), example (14a) is a doubly embedded S-O raising construction. The collocation of the raised objects in one surface position, as well as the collocation of the infinitives in one surface position, cf. (15b), is accomplished by clause union, as explained in Section 5.3.

(15) a.



b.



The relative ordering of objects and verbs remains to be explained. Recall that a single position in a holder can be occupied by a list of several constituents. E.g., the position of the clause-final verb cluster contains an ordered list of verbs. The relative ordering of the verbs follows from a rule which specifies that, in Dutch, constituents are by default added to the *end* of the list. Thus, the cross-serial pattern emerges quite

automatically, since deeper embedded constituents are added later than shallower ones. For a language like German, which is quite similar but accumulates the verbs in the reverse order, the opposite rule which adds verbs to the *front* of the list is postulated (cf. also Kempen & Hoenkamp, 1987). Finally, the English equivalent is simply accounted for by the absence of clause union, so that the embedded clauses are retained in the c-structure. I conclude that the same f-structure can easily account for the different surface phenomena in the three languages mentioned.

5.5 Unbounded dependencies

Interrogatives in Dutch are characterized by a marked word-order. Yes-no questions, e.g. (16a) show subject-verb inversion. In Wh-questions, the interrogative pronoun is normally fronted (16b) although this is not necessary (16c). Wh-fronting in itself is not seen as a discontinuity in SG. However, certain verbs allow a Wh-constituent to escape from an embedded clause in order to be fronted in the matrix clause; this results in a discontinuity which I will call *Wh-extraction* (16d). Optional fronting and possible resulting discontinuities are also observable with focused elements (16e), which suggests that Wh-extraction and focus extraction can be treated in a similar fashion.

(16) a. Eet Otto een appel? (Does Otto eat an apple?)

b. Wat eet Otto? (What is Otto eating?)

c. Otto eet wat? (Otto eats what?)

d. Wat denk je dat Otto eet ? (What do you think Otto is eating?)

e. Een appel denk ik dat Otto eet. (An apple I think Otto is eating.)

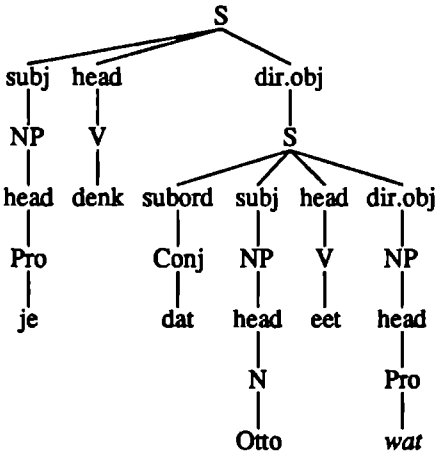
The treatment proposed below roughly follows the lines suggested by Kempen and Hoenkamp (1987) but works more incrementally and is extended to cover focus fronting as well; henceforth Wh-constituents will be considered focused. The discontinuities are accounted for in terms of a non-isomorphism between corresponding f-structures and c-structures. A treatment similar to that of clause union may cause an embedded constituent to be 'raised' to a higher level in the c-structure, where it occupies the clause-initial position, which is reserved for focused constituents. Let us now have a closer look at this process.

As with other word order variations, the temporal properties of the generation process are considered primarily responsible for the marked word order of focused constituents. I assume that those parts of the semantic input which are to be realized as focused constituents, are passed to the Formulator at a very early stage in the sentence generation process. This causes them to occupy a sentence-initial position. However, if for some reason the focused semantic elements are not accessible in time, the Grammatical Encoder may already have assigned another constituent a position in the

first holder slot. This may result in a question with unmarked ('declarative') word order as in (16c)[3].

For sentences involving Wh-extraction (16d) and focus extraction (16e), the same timing factor is important, but some additional machinery is necessary to allow the extraction of a constituent from an embedded clause to a higher level clause. This possibility must be indicated at the level of the lexical entry. Unlike clause union, where the destination of *all* constituents of a clause is transferred to a higher level, we must be more selective now. A special feature called *focus-destination*[4] is proposed, which handles the destination of focused elements. A focused element will first attempt to occupy a specified spot (position 1) in the holder of its *focus-destination*, otherwise it will go to its normal (default) destination. An analysis of (16d) is presented as f-structure (17a) and c-structure (17b).

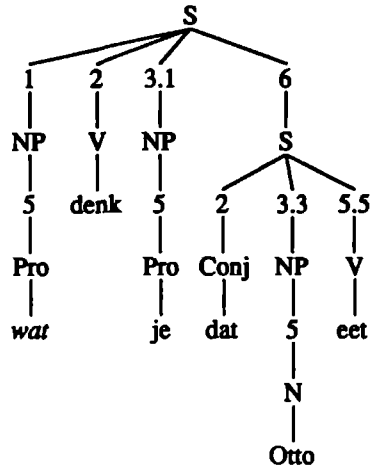
(17) a.



[3] Kempen and Hoenkamp (1987:233) account for 'declarative' word order in questions by assuming that Wh-fronting occurs only in the presence of a special ?X tag. The treatment proposed in the current work does not rely on this special tag but exploits the incremental assignment of word order to choose between alternatives.

[4] This feature takes the role of the variable WH-DEST in IPG (Kempen & Hoenkamp, 1987).

b.



In lexical entries which allow focus extraction (including Wh-extraction), this feature is specified on the segment for the object complement clause which is associated with the entry. If the lexical entry allows focus extraction, e.g. Dutch *zeggen* (to say) or *zien* (to see), then the feature in the foot node of the segment S-direct-object-S will refer to the feature in the root node. This is schematically shown in Figure 5.2. If the lexical entry does not allow focus extraction, e.g. Dutch *weten* (to know), then the feature will be absent.

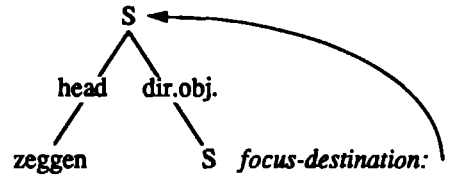


Figure 5.2
Dutch ***zeggen*** (to say) is a lexical entry allowing focus extraction.

In a fashion similar to the treatment of clause union, the value of the feature *focus-destination* may recursively refer upward in multiple embedded clauses. A remaining question is then, when and how to stop. In declarative clauses and direct questions, the final destination for focused elements is clearly the main clause. However, the mechanism should also work for indirect questions, e.g. (18a,b), where the final destination of the focused element is a subordinate clause.

(18) a. Ik weet wat je ziet dat Otto eet.
(I know what you see Otto is eating)

b. Ik weet dat je ziet wat Otto eet.
(I know that you see what Otto is eating)

It seems to be necessary for the Grammatical Encoder to know exactly which clause is being questioned. This can be indicated by means of a feature *interrogative* on the S in question. I assume that this feature has been set as a consequence of processing the semantic input structure. When such a feature is present, the *focus-destination* of an S refers to that S itself rather than upward.

5.6 Pronominal adverbs

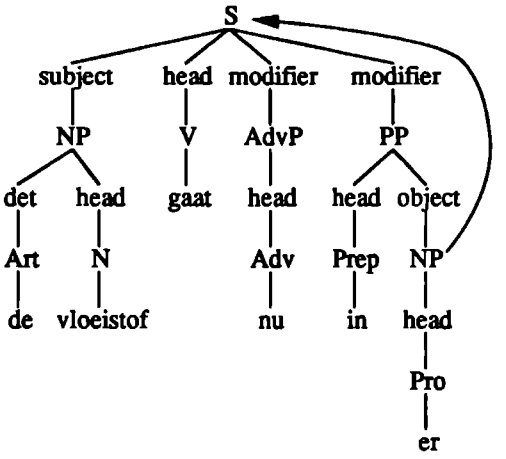
The Dutch adverbs *er*, *daar* (there) and *hier* (here) sometimes serve as variants on the pronouns *het* (it), *dat* (that) and *dit* (this) respectively, because the latter pronouns are not tolerated by many prepositions, e.g. (19a). This use of pronominal adverbs may result in a discontinuity of the prepositional phrase (cf. 19b).

(19) a. *De vloeistof gaat nu in het. (The liquid now goes into it)

b. De vloeistof gaat er nu in. (id.)

Apparently *er* is part of the S at the level of c-structure and thus it must be allowed to escape from the PP. Its destination is not the default, i.e., its mother node in the f-structure, but the next higher node. This is a property of the lexical entries for the pronominal adverbs; so this exception does not interfere with the general mechanism. F-structure (20) shows the destination of the pronominal adverb in example (19b). The S holder uses slot 3.5 for this constituent.

(20)



There are constraints on the number of occurrences of *er* in the same c-structure clause. E.g., suppose that the sentence gets an indefinite subject and is initiated by means of the 'situating' *er*, as in (21a,b), then if a pronominal adverb *er* is present, one of the occurrences of *er* must be omitted (21b). I have not studied this phenomenon.

(21) a. Er gaat een vloeistof in de fles. (There goes a liquid into the bottle)

b. Er gaat een vloeistof in. (There goes a liquid into [it])

5.7 Concluding remarks

I have presented discontinuities not only from a structural viewpoint but also from a processing viewpoint. Several kinds of discontinuities seem to offer advantages for an incremental strategy in sentence generation. This holds especially for the optional dislocations. Right dislocation allows the generator to utter constituents which are ready, and to postpone uttering more complex ('heavy') ones, which are still being processed, to a later stage. In addition, right dislocation allows the incorporation of new semantic input as afterthoughts. Fronting of focused constituents is also natural in an incremental mode of generation, if we assume that prominent concepts are passed on to the Grammatical Encoder earlier than other parts of the semantic input. In contrast, S-O raising benefits less from an incremental mode because it seems to require planning ahead.

True discontinuities in SG are viewed as differences between ID relations in c-structures and those in corresponding f-structures. Constructions which are treated in this way include clause union, right dislocation, and fronting. Separable parts of words such as verbs and compound prepositions are not viewed as true discontinuities but have their origin in lexical entries consisting of multiple segments.

The use of c-structures in SG is somewhat similar to LFG, but contrasts with other approaches such as DPSG which are based on PS grammar. Whereas DPSG attempts to fit both functional relations and surface constituency into one structure, SG distinguishes between an unordered functional structure and an ordered constituent (surface) structure.

I make the following tentative generalizations about SG mechanisms for discontinuities. The destination of a constituent must always be a node which dominates it—but not necessarily immediately. There seem to be two major variants of the destination mechanism allowing constituents to go to non-immediately dominating destinations. The first is *root*-initiated. In these cases, e.g. clause union, a node refers its constituents to another dominating node. This operation may be recursive. The second mechanism is *foot*-initiated. In these cases, e.g., PPs with pronominal adverbs, a node directly presents itself to a higher-level destination.

Part Two: Representational and computational aspects

Part Two opens with Chapter 6, which is concerned with the question, how to devise a suitable computational framework which represents linguistic concepts in an efficient and flexible way. In response to this question, an *object-oriented* representation of linguistic knowledge is introduced. A grammar formalism can be called object-oriented if its important linguistic units (such as words, phonemes, and phrases) are modeled as active computational units which share knowledge by default inheritance. By means of some concrete examples, it is shown how various inheritance mechanisms account for regularities in language, avoid redundancy, and allow the elegant extension of general knowledge with exceptions. CommonORBIT is presented as a suitable language for an object-oriented representation of linguistic knowledge.

Chapter 7 describes how an object-oriented grammar dovetails into a lexicalist approach to grammar. A review is given of default inheritance in other linguistic models, ranging from early AI approaches to several recent theories of language.

Chapter 8 describes an object-oriented implementation of Segment Grammar (see Chapter 3). The various linguistic units involved in this formalism—nodes, features, and segments—can be represented as objects, which allows their representation to take advantage of inheritance.

Chapter 9 probes the potential of *parallel processing* in incremental sentence generation. It describes how IPF sets up parallel processes to execute tasks simultaneously with other ones. Finally, a parallel distributed unification mechanism is proposed in order to allow the simultaneous construction of distinct parts of one syntactic structure.

6 A framework for the representation of linguistic knowledge

Newell (1981) views knowledge as a *competence*-like notion. He makes an explicit comparison between knowledge in AI, which is a potential for generating intelligent solutions, and Chomsky's use of the term *competence* as 'knowledge of language', which is a potential for generating speech. Newell presents the *knowledge level* hypothesis: there exists a distinct level in computer systems which is characterized by knowledge as the medium and the principle of rationality as the law of behavior. The knowledge level consists of knowledge *and* an access structure to that knowledge. It contains no specification of processing mechanisms, only a global principle of rationality to be satisfied by the system behavior.

The development of knowledge representation languages has been recognized as a fundamental and profound contribution to cognitive science in general. But it is also a contribution to a cognitive approach to linguistics in particular. Newell's comparison between the *knowledge level* in AI and *linguistic competence* could be interpreted as a suggestion that the right level for the expression of linguistic competence is the level of knowledge representation. Although Newell probably did not intend this, the idea is nevertheless the rationale underlying this chapter.

6.1 Introduction

6.1.1 Why representation languages?

It is sometimes argued that since every computable process can be represented as a computation on a Turing machine, there is no point in developing a new representation paradigm. This argument is invalid because we do not design languages in order to extend the class of computations they can perform. Rather, what we do in defining a knowledge representation language is to make an abstraction over a class of representational structures and introduce a syntactic mechanism to express that abstraction. The resulting new primitives will manage the complexity of knowledge so that programs will be more understandable, modularity will improve, etc. Thus there is a practical gain. But the relevance of postulating new representational primitives goes beyond mere productivity concerns. They make an empirical statement by stating generalizations about representational structures used by processes requiring intelligence. As with all empirical generalizations, it may be possible to find cases demonstrating why a generalization is appropriate, without it being possible to prove

that it is adequate or valid. However, we can try to falsify a representation language by showing that its primitives are an obstacle to system development. Such (partial) falsification efforts can lead to the construction of a better representation language (cf. Steels, 1978a).

Kempen and Hoenkamp (1987; Hoenkamp, 1983) propose a procedural representation of linguistic knowledge. Procedural knowledge is knowledge about *how* to do things. In a procedural representation, there is no general inference mechanism which is carefully kept separate from a database of propositions or rules. Rather, a procedural representation is a program to carry out a task; it has a specific control structure for doing so. A procedural representation therefore seems often useful for the simulation of processes which require no explicit reasoning. An example of such 'automatic behavior' is the use of linguistic knowledge: people process language in various ways (e.g. production, parsing, translation), yet they can rarely explain the knowledge involved in these activities.

One way of expressing knowledge in a procedural representation is to view structural elements in the problem domain as active procedures which each have their own control structure. For example, in the procedural grammar proposed by Kempen and Hoenkamp, syntactic constituents and functions are viewed as active procedures, and a syntactic tree structure is viewed as a representation of a hierarchy of subprocedure calls. The top of the tree is the main program; lower nodes are successive subprocedures. A fully consistent representation of linguistic units as procedures opens up a number of novel possibilities. Kempen and Hoenkamp propose the execution of 'sister' branches in parallel and the representation of coordination as a kind of iteration.

However, a procedural representation has no special provisions to represent *generalization* and *specialization* in a hierarchy of linguistic concepts. Hierarchical reasoning is necessary for a practical and realistic representation of linguistic knowledge. The development of extensible and adaptable natural language processing systems depends on a knowledge representation paradigm within which generalizations are effectively exploited (Jacobs, 1985). But also from the psychological perspective, abstraction is important. People can generalize and specialize, classify with respect to abstract categories, form prototypes, and make use of defaults. Finally, many traditional grammars are organized by means of abstraction over linguistic categories. It is therefore desirable to devise a framework for the representation of linguistic knowledge which incorporates primitives for hierarchical reasoning.

6.1.2 Object-oriented knowledge representation

Representations based on *structured objects* offer primitives to express generalizations over representational structures. Fikes and Kehler point out that

> "they capture the way experts typically think about much of their knowledge, provide a concise structural representation of useful relations, and support a concise definition-by-specialization technique that is easy for most domain experts to use." (Fikes & Kehler, 1985)

In the object-oriented programming paradigm, knowledge is grouped in manageable symbolic units—objects. Objects may represent situations, concrete things, or abstract categories. In the domain of medical diagnosis, some of the objects involved are patients, diseases, bacteria, and symptoms. In the domain of natural language, some of the objects are sentences, phrases, words, and speech sounds.

Objects are structured: each object has a number of *aspects* which contain knowledge relevant to the object. E.g., a patient has the aspects *current-symptoms, medical-history, blood-pressure,* etc. In addition, a patient has the same aspects which are relevant for each normal person, i.e., *name, sex, weight, age,* etc. Clearly, some of these aspects, e.g. *name,* contain static knowledge and could be stored as in a data base. But other aspects, e.g. *blood-pressure,* are dynamic and may continuously change during a computer simulation. Some aspects can be computed from other knowledge, e.g., the *age* of a person at a certain time can be computed from the person's *date-of-birth.*

In the linguistic domain, there could e.g. be an object representing the vowel /i/ which has aspects representing its phonological features. E.g., it has an aspect *rounded* with value *0*, an aspect *front* with value *1*, etc. In addition, it inherits a number of aspects from *vowel*, such as *consonantal, syllabic*, etc. With each phonological segment, a program could be associated to drive an articulatory apparatus.

Object-oriented languages provide a direct means for the representation and manipulation of such objects and provide access to their aspects. Computational objects can be created and changed in the course of the computation. They are separate units, each one containing all the knowledge—procedures as well as data—which is relevant to an entity in the real world. This knowledge forms the internal state of an object, which can be modified during the computation. Thus, objects may represent entities which change over time and which interact in various ways.

An important feature of object-oriented languages is that they provide some kind of mechanism for objects to share their structure and behavior with other ones. Sharing is usually one-way (to avoid circularity): an object can inherit knowledge associated with another one. The goals of knowledge sharing can be seen from different points of view:

1 *Specialization.* From a conceptual point of view, knowledge sharing mechanisms allow subtyping in the form of specializations of a general object. In this way a *specialization hierarchy* is produced, which corresponds closely to the hierarchy of *'is a'* relations in a semantic network. E.g., *vowel* and *consonant* might be subtypes of *phonological segment.* A hierarchy of grammar concepts might contain objects for *word* and its subtypes *noun, verb, preposition,* etc.

2 *Combination.* Another use of knowledge sharing is the representation of an object as a combination of two or more other objects. This kind of knowledge sharing is often called *multiple inheritance.* This is often done if an object needs to integrate knowledge from different sources or perspectives. E.g., *John* may be a male patient; thus the behavior of *man* and that of *patient* is combined. Composition will often consist of the addition of a few special features (e.g. *patient*) to a more general-purpose category (e.g. *man*); the object whose features are 'added' is sometimes called a *mixin.* A *transitive*

compound strong verb could be defined as a combination of *transitive, compound,* and *strong verb.*

3 *Stepwise refinement.* A program can be constructed by first modeling the most general concepts in the application domain, and then dealing with special cases through more specialized objects. The programmer is not so much concerned here with the construction of a taxonomy but rather with refinement as a programming methodology. Stepwise refinement by specialization can be compared with the well-known methodology of stepwise refinement by decomposition (Wirth, 1971). It is significant that the effort of defining an object is *proportional* to the extent in which it differs from other objects. Thus refinement is not only useful as a programming methodology, but it can also be thought of as a general cognitive mechanism, for it reflects a *principle of least effort.*

4 *Avoiding redundancy.* From the point of view of data storage, knowledge sharing provides efficiency by avoiding redundancy. A piece of information which is necessary in many objects needs to be stored in only one object. This not only reduces the memory needed to store a piece of knowledge, but also improves modularity because the shared knowledge needs to be updated only in one place. Again, this is thought of as a general cognitive principle and not just a software engineering strategy.

It is important to mention that inheritance is based on *defaults* rather than absolute and irretractable statements. If we say "birds fly" then we mean "birds typically fly" and we have no problems to accept and handle exceptions—e.g. ostriches and penguins—adequately. Similarly, if we say "nouns are countable" then there may still be a special class of uncountable nouns. Exceptions cannot be handled well in first order logic, because to derive for a particular bird that it can fly, it would be necessary to first prove that it is not exceptional, i.e., that it is *not* an ostrich or a penguin, etc. Rather, we want to be able to derive that a bird can fly if it *cannot* be shown that it is exceptional (cf. Brewka, 1989). In object-oriented languages, inheritance is based on this kind of reasoning—default reasoning. All inherited knowledge only holds in so far as it is not overruled by knowledge in the inheriting object. From a philosophical viewpoint, one might say that default inheritance gives the old philosophical *ceteris paribus* idea a computational counterpart.

Object-oriented programming belongs to a family of knowledge representation languages including *frames* and *semantic networks.* The formalisms in this family all attempt to group knowledge in somewhat larger units than, e.g., production rules. Semantic networks represent entities in memory as nodes, and associative relations between entities as arcs which link nodes. Arcs labeled *'is a'* or *'a kind of'* represent specialization relations which allow the inheritance of properties (Brachman, 1979). Frames have somewhat lost the original meaning intended by Minsky (1975). In many frame-based languages, frames correspond closely to structured objects. The different aspects of knowledge about an object are usually called *slots* in frame terminology. In some frame systems, as in most semantic networks, slots contain only declarative knowledge (slot *fillers*). The inclusion of procedural knowledge in frame slots (which is standard in object-oriented languages) is called *procedural attachment* (Bobrow & Winograd, 1977).

6.1.3 Towards an object-oriented grammar

The adoption of an object-oriented framework for the representation of linguistic knowledge amounts to the conception of an *object-oriented grammar*. Obviously, it is not appropriate for a grammar to be be called truly object-oriented simply because it is implemented in a particular programming language. The following definition is proposed:

> *A grammar is* object-oriented *if its important linguistic units are modeled as active structured computational units, and linguistic regularities are expressed by means of default inheritance.*

This definition is purposely not very restrictive. The most obvious object-oriented approach to grammar consists of the representation of semantic, syntactic, morphological and phonological *categories* as objects. Since this study is concerned with grammatical encoding, syntactic and morphological categories will primarily concern us in this chapter. Chapter 8 will extend this approach to syntactic segments. Thus the lexicon becomes the basic framework where all grammar knowledge is attached to. However, other object-oriented approaches are also possible, e.g., the representation of *rules* and language *processes* as objects. E.g., in a transformational framework, there could be a taxonomy of transformations. A full discussion of these possibilities is beyond the scope of this work.

Some theories of language view linguistic units as active computational agents but do not fit well into the object-oriented paradigm. A typical system of this kind is Word Expert Parsing (WEP; Small, 1983; Adriaens, 1986), where words are viewed as 'experts' which trigger processes that idiosyncratically control the parsing process. WEP is object-oriented in the sense that active lexical agents cooperate to comprehend natural language. However, WEP seems to be based on a procedural representation: *word experts* are coroutines which temporarily control the whole parsing process and then terminate or suspend themselves (passing on control to another word expert). Moreover, word experts do not use the full range of capabilities of an object-oriented representation. In particular, no use of inheritance is reported.

In a similar vein, the object-oriented parser proposed by Phillips (1984) uses active computational units which operate in parallel to analyze a natural language sentence, but apparently there is no exploitation of the inheritance mechanism to capture regularities in the grammar. Phillips' parser is based on an augmented phrase structure system and implemented in FLAVORS (Weinreb & Moon, 1980). His system differs from WEP in that the knowledge is grouped around *rules* rather than words: a *constituent* flavor creates an object that is associated with a rule of the grammar. A scheduler takes in a word of input and delivers it to instances of *constituent*; each constituent instance basically tries to advance its state by matching the current input against the tail of the rule. In doing so, more instances of rules may be created and messages may be sent to other rules. By the way, the conceptual knowledge base of Phillips' system is also object-oriented: it is a semantic network with objects as nodes; arcs designate neighboring objects and the kinds of message that can be sent to them.

I will now turn my attention to approaches in which the use of default inheritance in the representation of linguistic *categories* is a central issue.

6.2 Inheritance in linguistic representation

This section will present some arguments for the use of default inheritance. It will do so by means of examples rather than by presenting a formal apparatus. It will be indicated how various methods of using inheritance can represent and exploit linguistic regularities and how the use of defaults can elegantly handle exceptions. This brief overview is naturally incomplete, but merely exemplary. At the same time it will become clear that the expressive power of the object-oriented paradigm itself puts almost no restrictions on the way knowledge can be defined, organized and used. Different alternative hierarchies can be considered to account for the same phenomena. A principle of maximum economy could serve as a criterion for the evaluation of alternative organizations.

6.2.1 Prototypes vs. classes

Many object-oriented systems use *classes* (or *types, flavors*) as a means of data abstraction. A class can be viewed as a set of a number of objects (*instances*) sharing some common behavior. Linguistic examples of classes are notions like *noun phrase, adjective, strong verb, phoneme*, etc. An *is a* relation holds between a class and its instances. One then says, for example, that *blue* is an adjective. Since a class is much like a *set*, the relation between a class and its instances can be represented graphically as in Figure 6.1.

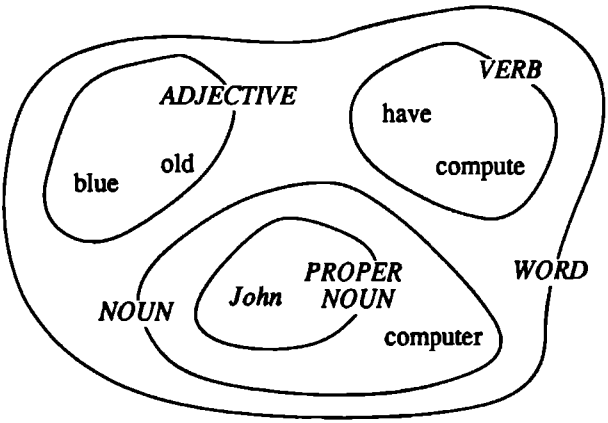


Figure 6.1
An example configuration of linguistic classes and instances

However, there is also another, less common mechanism of behavior sharing based on the notion of analogy rather than on classification. An existing object can sometimes

be seen as a *prototype* (or *stereotype, model, proxy*) for another (Lieberman, 1986). Between such objects, an *is like* relation (or *conformance* relation) holds. *Is like* relations are not uncommon in some traditional grammars, especially for the classical languages, which do not introduce classes but promote one member of the class as an example for all others. E.g., the Latin noun *rosa* is often promoted as the stereotype for a certain kind of nouns. We then say, e.g., that *porta* "is like" *rosa*. In this way, regularities are expressed by a specification of how a lexical entry differs from another, similar one. This example is depicted in Figure 6.2.

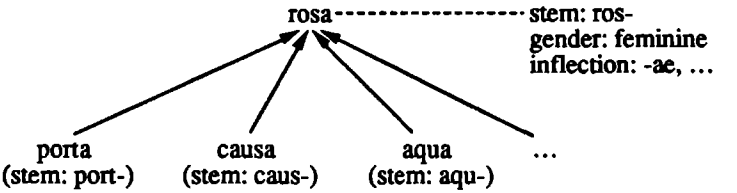


Figure 6.2
Inheritance from a stereotypical lexical entry rather than a class

Information belonging to an inflectional paradigm is thus attached to one prototypical lexical entry. Clearly, this is a more economic way of encoding paradigmatic relations, because it dispenses with special objects for classes. It seems that linguistic knowledge is likely to be acquired in this way, either subconsciously or by teaching. Still, I will often follow the main linguistic tradition and use classes in this study.

Without claiming to fully answer the empirical problem how people learn and represent paradigmatic relations, some properties of object-oriented representation can be pointed out. The modular nature of object-oriented systems allows the easy addition of new objects without necessarily having to revise existing ones. As one learns new words, they can easily be defined as instances of a class or specializations of a stereotype. The effort of defining a new object is proportional to the extent in which it differs from other existing objects. This representational generalization is relevant if we assume that there is a principle of least effort in the cognitive system.

Assuming that language learners are able to make abstractions of the data which they acquire, it is likely that tentative hierarchies are sometimes revised. In a modular object-oriented representation, it is easy to change the prototypes of an object. It seems that children sometimes assign new words to the wrong class, e.g., Dutch children sometimes inflect a weak verb as a strong verb (or *vice versa*), or even inflect an adverb as a verb. When there is enough evidence to make a revision, the prototypes are changed.

The acquisition of more general classes (superclasses) could be modeled by filtering out common behavior from a number of objects and storing it in a new object from which these objects then become specializations. New members of a class will acquire the same common behavior. More specialized objects develop by *foliation*: the differences with existing objects are determined and subsequently the new object is created, or *reified*. However, little is known about how people actually manage

regularities. Thus the representations presented here remain mere proposals for cognitive architectures, with evidence coming more from regularities in the language itself than from observed language behavior.

6.2.2 The importance of uniformity

Hierarchies consisting of specialization relations can be used to organize linguistic knowledge. An object-oriented representation of linguistic categories is advocated by De Smedt (1984). I will follow the example given there. Consider, e.g., the hierarchy in Figure 6.3, which accounts for some regularities in the inflectional morphology of Dutch verbs.

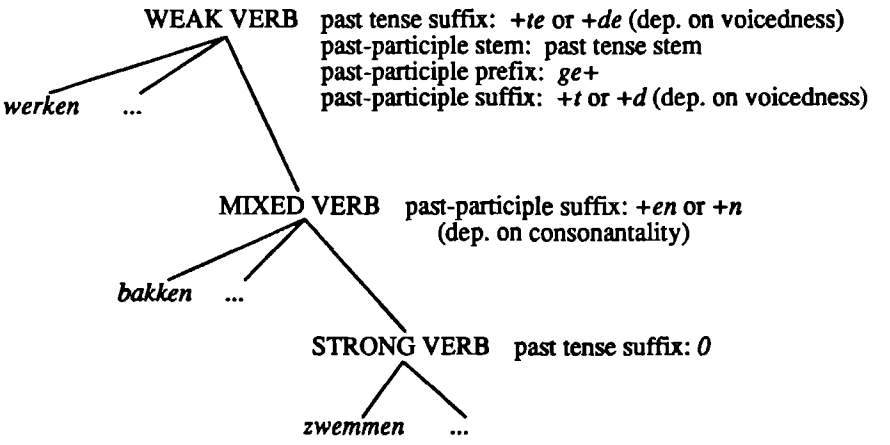


Figure 6.3
A partial hierarchy of Dutch verbs (from De Smedt, 1984)

The top node of this hierarchy, *weak verb*, is the most regular kind of verb and therefore is the prototype object for all other verbs. Actually, this object inherits from an even more general object, *word*, and so on. The inflectional behavior of the weak verb is represented in a number of aspects which each compute an inflectional form from a stem, prefix, and suffix. All weak verbs have the object *weak verb* as a prototype. E.g., *werken* (to work) is an object which inherits its inflectional behavior directly from *weak verb*. There is a specialization representing a subclass, *mixed verb*, which is partly regular but has an exceptional past participle suffix. This contradicts the specific information associated with its prototype for this particular form, which is thus overruled. Again, there are a number of verbs which inherit their morphological behavior from *mixed verb*, there is a subclass, and so on. This representation provides a non-redundant and generalization-capturing account of Dutch verb inflections.

It is significant that the hierarchy in Figure 6.3 contains lexical items as well as morphological categories. Thus, there is no strict separation between the lexicon and the body of morphological knowledge. Instead, there is a smooth transition from the most general categories to the most specific ones—individual words. This is a direct

consequence of the *uniform* representation of linguistic knowledge as objects. A uniform framework allows for the same general principles to be applied to a variety of knowledge forms (Jacobs, 1985).

6.2.3 Multiple inheritance 1: selective aspect inheritance

Specialization is but one facet of the inheritance mechanism. The other is the formation of new objects as a composition of several prototypes. This presupposes the handling of multiple inheritance relations. I will avoid the many thorny issues in multiple inheritance (e.g. Touretzky, Horty & Thomason, 1987), but nevertheless I must raise the question of how conflicts between contradicting knowledge in the composing objects can be resolved.

One special inheritance mechanism to solve this problem is *selective aspect inheritance*, which means that the programmer explicitly selects which aspects are inherited from which prototypes. Suppose that on independent grounds we want to reorganize the hierarchy in Figure 6.3 so that the weak verb is not necessarily the most stereotypical kind of verb. Rather, the most general kind of verb has three subclasses, *weak verb*, *strong verb*, and *half-strong verb* (i.e., *mixed verb*), which are on the same level in the hierarchy depicted in Figure 6.3.

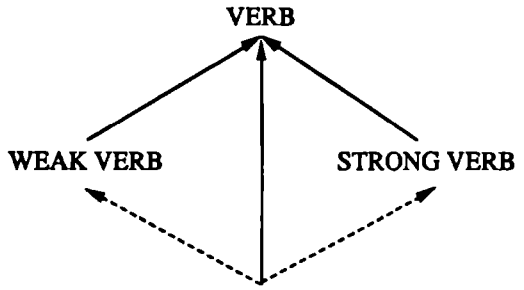


Figure 6.4
Alternative hierarchy of Dutch verbs using selective inheritance

The behavior of *half-strong verb* is partly obtained from two 'sister' classes. It is necessary to indicate exactly which aspects are inherited from which objects, because a simple ordering of prototypes would not suffice: the past tense is weak, but the past participle is strong.

Daelemans (1987a) exploits selective aspect inheritance for morphology using the KRS system. Although selective aspect inheritance is very powerful, the hierarchy as a whole is less motivated because it involves more objects and more inheritance links while the objects are on average less motivated because they provide fewer defaults for other objects. Thus, the gradual specialization in Figure 6.3 seems preferable.

6.2.4 Multiple inheritance 2: specialization by composition

Another way to resolve conflicts between composing objects is to establish an explicit priority among objects. E.g., suppose that *rondrijden* (to ride about, to tour), a Dutch compound verb, inherits from both *compound* and *verb*. Suppose, furthermore, that we specify that the knowledge in *compound* has priority over that in *verb*. Any knowledge in *compound* will precede that in *verb*. In principle, we could implement this as a depth-first search in the hierarchy (left-to-right in Figure 6.5).

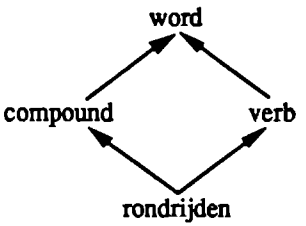


Figure 6.5
Multiple inheritance with a common prototype

However, suppose that they both have a common prototype, e.g. the object *word* in Figure 6.5. A depth-first search in the hierarchy will reflect the priority relation between *compound* and *verb*, but will not do justice to the specialization relations. The common prototype must not be considered before more specialized objects in the hierarchy. This can be formulated as the following principle, which has also been advocated by Ducournau and Habib (1987):

(1) *Specialization vs. Multiplicity:*

Inheritance must follow the specialization partial order; therefore, in any case the specialization relation excels the multiplicity relation.

An example in the domain of the morphology of verbs, which shows the use of this principle, will be given now. In Dutch, there are in fact more classes of strong and half-strong verbs than those dealt with so far. Some half-strong verbs are like the first class, but have a vowel change in the past participle; this kind will be called *half-strong verb 2*. A third class is partially weak and partially strong, but in exactly the opposite way: it has a strong past tense (with vowel change), but a weak past participle; let us call this kind *half-strong verb 3*. The new hierarchy is depicted in Figure 6.6.

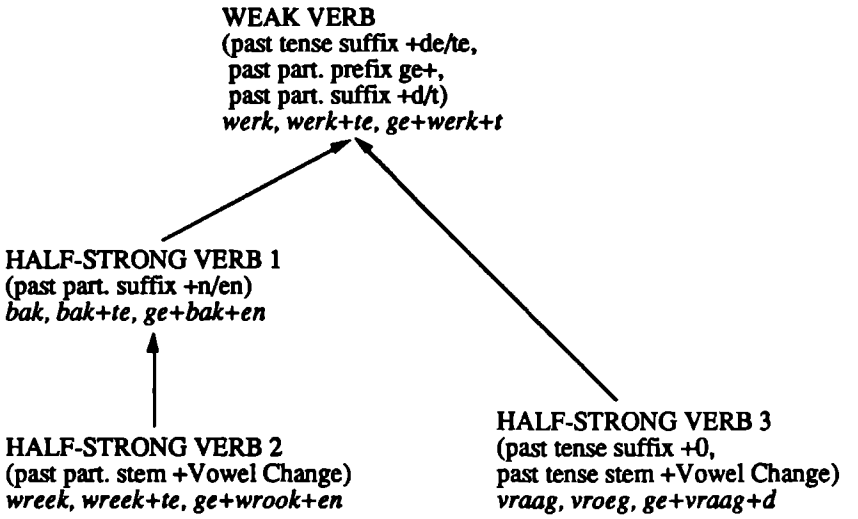


Figure 6.6
Objects for Dutch half-strong verbs

The strong verbs in Dutch can also be divided into two classes, one which exhibits a vowel change in the past participle and one which does not. This offers the opportunity to use multiple inheritance, because the behavior of the strong verbs can be completely composed out of the half-strong verbs, as shown in Figure 6.7. This representation is especially powerful because the definitions for the strong verbs do not need any specification of exceptions and can thus be extremely simple (see Section 6.3.1). The multiple inheritance principle (1) makes sure that more specialized prototypes have priority over more general ones. Thus, e.g., *strong verb 1* inherits from both *half-strong verb 1* and from *half-strong verb 3* before the more general knowledge in *weak verb* is considered. Thus, the defaults in *weak verb* are effectively blocked. In this hierarchy, the left-to-right priorities are irrelevant because the main classes of half-strong verbs are opposite and thus complementary; they each provide differing specific information which does not contradict each other. Principle (1), however, is crucial for correct inheritance.

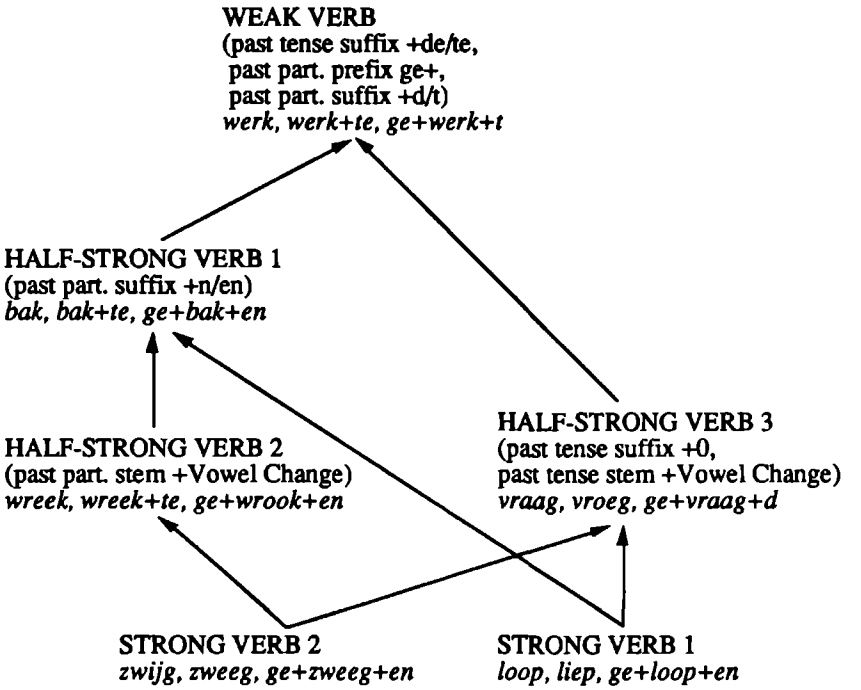


Figure 6.7
Revised hierarchy of Dutch verbs[1]

6.2.5 Multiple inheritance 3: mixins

The previous section illustrated the composition of an object out of complementary but comparable prototypes. A different use of multiple inheritance consists of the composition of knowledge from various kinds of sources. Often, one of the prototypes will designate a basic category and the others will be *mixins* which add some extra information. In an English description of such composition, one will often use an adjective for the mixin, e.g. *transitive* verb, *plural* verb, *compound* noun, *nominative* noun, etc.

In the case of a *nominative noun*, an object is composed out of a feature and a category. E.g., suppose that an NP, a noun, a determiner, etc. do not by default have the feature *nominative*. Sometimes we want to create a *nominative* NP, a *nominative* noun, a *nominative* determiner, etc. The feature *nominative* can be factored out in the form of a separate object which can be added as a mixin to the categories NP, noun, determiner, etc. Multiple inheritance is useful here because the alternative, the creation of a dedicated new object *nominative-NP* representing the class of nominative NPs, another

[1] This hierarchy is partially due to Walter Daelemans (1987a:52), who nevertheless uses selective aspect inheritance rather than multiple inheritance.

one representing the class of nominative nouns, etc., would cause an unneccessary proliferation of classes. Left-to-right priorities are important when the mixin is meant to override any defaults in the remaining prototypes. An example is given in Figure 6.8.

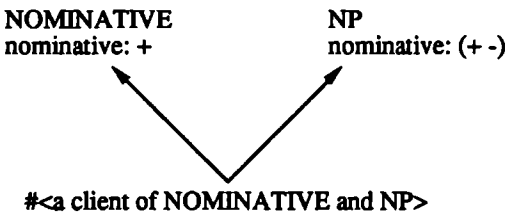


Figure 6.8
A feature *mixin* relies on explicit order in multiple inheritance

Multiple inheritance can also be used to factor out morpho-phonological and semantico-syntactic aspects in lexical rules, which was proposed by Jackendoff (1975). The combination of these different knowledge sources is used in the HPSG subcategorization system which is discussed below, e.g. a *transitive finite verb* is composed by adding the mixins *transitive* and *finite* to *verb* (see also Section 7.3.2). Another use of multiple inheritance in lexical rules is the addition of new lexical categories in a derivation. E.g., a past participle of a verb may be used as an adjective by assigning it the prototype *adjective* in addition to *verb*. This use of inheritance also relies on explicit order: the object now inherits its morphological behavior of an adjective.

6.2.6 Exceptions and 'blocking' in structured objects

As already mentioned in Section 6.1, inheritance is based on the notion of *defaults*. Using default theory, we can handle exceptions without having to revise existing general knowledge. This mechanism can be used to model *blocking* of general word forms when an exceptional form is present. E.g., suppose that we want to account for the fact that in Dutch verbs with a derivational prefix, the usual past participle prefix does not occur. E.g., whereas the past participle of *werk+en* (to work) is *ge+werk+t*, that of *ver+werk+en* (to process) is *ver+werk+t*. The object *verwerken* could be derived from *werken* by means of a lexical rule. We can arrive at the most efficient description by letting the word *verwerken* be a specialization of *werken*. It differs minimally from its prototype by the fact that its prefix is *ver+*, as shown in Figure 6.9.

Figure 6.9
Inheritance in derivation by prefixing

The morphological components of verbs—prefix, stem and suffix—can be modeled as aspects of verb objects. To obtain the present tense plural, *ver+werk+en*, the prefix, stem and suffix are concatenated.

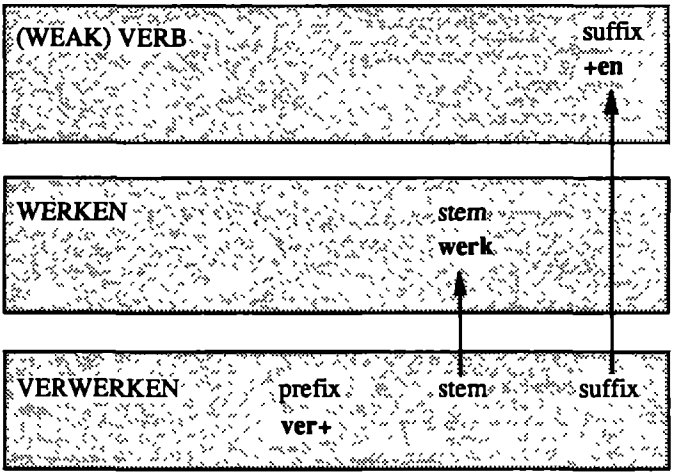


Figure 6.10
Default inheritance in structured objects: present tense plural of *verwerken*

Each piece of knowledge is represented in the most general linguistic object possible. The present tense plural suffix holds for weak verbs and all objects that inherit from it, the stem *werk* holds for the verb *werken* and all objects that inherit from it, and the prefix is specific for *verwerken*.

To obtain the past participle in a traditional formalism, we would need a blocking rule to prevent the addition of the normal past participle prefix *ge+* (see Figure 6.11). However, an explicit blocking rule is unnecessary if we use default inheritance: the presence of a prefix in an object automatically prevents the inheritance of the default prefix in a prototype.

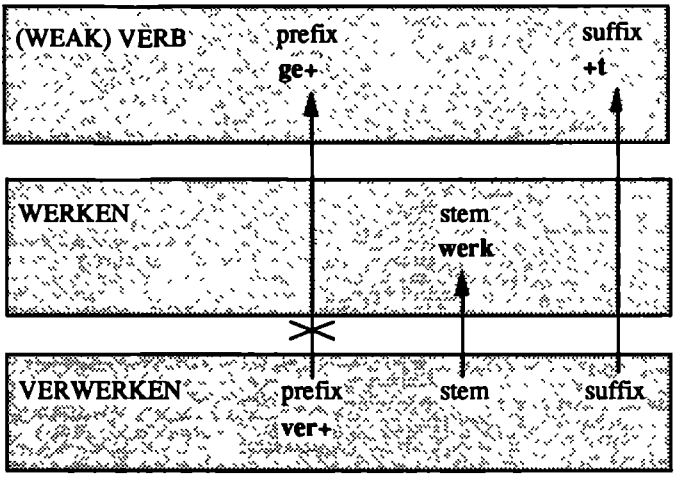


Figure 6.11
Default inheritance automatically takes care of blocking: past participle of *verwerken*

The default inheritance mechanism will delegate aspects to the prototype unless they are defined in the inheriting object. To form the past participle, the *stem* and *suffix* of *verwerken* are retrieved by inheritance, but not the *prefix*, because it is already present.

6.3 CommonORBIT

CommonORBIT (De Smedt, 1987; 1989) is an object-oriented programming language written as an extension of Common LISP (Steele, 1984)[2]. It is a further development of the ORBIT language which was proposed by Steels (1983). CommonORBIT is proposed here as a suitable and practical implementation language for an object-oriented grammar. I will not attempt here to provide a description of the full power of the language, but will concentrate on examples of the representation of linguistic knowledge.

6.3.1 Defining objects in CommonORBIT

Although CommonORBIT is based on prototype (or stereotype) relations, it can also represent a class system. A class can easily be modeled by designating an object as a class for other ones, which then inherit from it as from a prototype[3]. CommonORBIT

[2] For an introduction to LISP, the reader is referred to Winston and Horn (1988) or Anderson, Corbett and Reiser (1988).

[3] Systems based on classes are less general, because a distinction between classes and instances imposes restrictions on inheritance. In particular, objects cannot inherit from other objects, but only from classes. Also, a class is not an object and cannot itself perform any operations; classes have no use except for creating instances. Some languages, e.g. SIMULA, do not even allow the creation of classes while the program is running; classes are defined at compile time and remain fixed thereafter.

provides one general specialization relation which allows objects to inherit knowledge from other ones. An object can be defined by means of DEFOBJECT, which accepts the name of a new object followed by its prototypes and aspect definitions. E.g. one can write:

```
(DEFOBJECT PORTA ROSA)
```

to indicate that *porta* inherits from *rosa*. Now, this specification does not mean much unless it is indicated how an object differs from its prototype. So, e.g., we might add a definition of the aspect stem to indicate that the stem of *porta* is "port-":

```
(DEFOBJECT PORTA ROSA
  (STEM "PORT-"))
```

Everything but *stem* is inherited from its prototype *rosa*, which presumably has knowledge about its inflectional behavior. To create a class object, one could assert, e.g.:

```
(DEFOBJECT PROPER-NOUN NOUN
  (ARTICLE? NIL))
```

to specify that *proper-noun* is a subcategory of the category *noun* and that it does not need an article. Presumably, there is also an aspect *article?* defined for *noun*, but since a definition is present in *proper-noun*, that default is not inherited.

The objects in the above examples have a name (which is a symbol). In addition, anonymous objects can be created in CommonORBIT by means of A or AN, e.g.:

```
(A PROPER-NOUN)
```

Anonymous objects are often useful to represent *tokens* of linguistic categories, whereas named objects are often reserved for *types*. They are also useful as objects embedded in other ones. Rather than simply assigning a spelling as the stem of a word, we can state, e.g., that the *stem* of the object *werken* (to work) is a *morpheme* with a pronunciation and a spelling:

```
(DEFOBJECT WERKEN WEAK-VERB
  (STEM (A MORPHEME
           (PRONUNCIATION "wErk")
           (SPELLING "WERK"))))
```

CommonORBIT allows multiple inheritance. The priorities among prototypes can be programmed by means of left-to-right order in the DEFOBJECT form. E.g., the multiple inheritance relation of *rondrijden* in Figure 6.5 is written as follows:

```
(DEFOBJECT RONDRIJDEN COMPOUND VERB)
```

Similarly, the nominative NP of Figure 6.8 can be created simply by writing:

```
(A NOMINATIVE NP)
```

These examples show that CommonORBIT has a syntax which resembles that of natural language, although it is based on LISP. The enhancement of a formal language with quasi natural language constructs improves human-computer interaction (e.g. De Smedt, 1980).

The multiple inheritance mechanism of CommonORBIT takes left-to-right priorities into account and is based on principle (1) described in Section 6.2.4. The algorithm used is similar to one presented by Ducournau and Habib (1987). Therefore, we can, e.g., define *strong verb 1* (see Figure 6.7) extremely simply as follows:

```
(DEFOBJECT STRONG-VERB-1
  HALF-STRONG-VERB-1 HALF-STRONG-VERB-3)
```

Selective aspect inheritance is also possible in CommonORBIT. The relations in Figure 6.4 can be programmed as follows:

```
(DEFOBJECT HALF-STRONG-VERB
  VERB
  (PAST-TENSE-SUFFIX :DELEGATE 'WEAK-VERB)
  (PAST-PARTICIPLE-SUFFIX :DELEGATE 'STRONG-VERB))
```

Using DELEGATE instead of rewriting the complete definition for *half-strong-verb* not only simplifies the program by storing the knowledge in only one place, but it also has the advantage that the program is more modular: if the definition of *past-tense-suffix* for *weak-verb* is changed, the change will automatically apply to *half-strong-verb*.

CommonORBIT has several other sophisticated facilities for representation using inheritance. An example of *structured inheritance* will be given in Section 8.3.3.

6.3.2 Generic functions

CommonORBIT is not purely declarative but integrates procedural knowledge in the object definitions. Objects thus become active computational units rather than passive stores of information. Even the generation algorithm itself can be distributed among objects. Linguistic processing is modeled as the application of *generic functions* to linguistic objects.

A *generic function* is a function whose definition depends on the type of its arguments. A object-oriented approach to generic functions can easily be achieved by distributing the definition among objects rather than keeping it together as a single body of code. Like a normal LISP function, a generic function is called on arguments, performs certain operations, and returns values. Unlike an ordinary function, the actual operation to be invoked is not stored in the function definition itself, but in the aspects of the objects that the function is applied to. Conceptually, generic functions are useful because they allow one function name to be used for a high-level operation which requires different work for different kinds of objects. For the caller, generic functions provide a simpler interface, because the differentiation is carried out automatically by the generic function rather than explicitly by the caller.

In CommonORBIT, generic functions provide a uniform interface for procedural as well as declarative knowledge contained in the aspects of an object. E.g., the aspect

past-participle suffix in Figure 6.3 can be seen as a function which is distributed over (at least) two objects; the definition is stored in their aspects. Function calls may be embedded. E.g., to retrieve the pronunciation of the stem of *werken* (see above), the function *pronunciation* is called on the result of calling the function *stem* on the argument *werken*. This returns the string "wErk":

```
(PRONUNCIATION (STEM 'WERKEN))
"wErk"
```

The last example involves a function which retrieves declarative knowledge, i.e., it simply returns a value. However, aspects may contain arbitrary procedures which are executed when they are called as generic functions. Suppose, e.g., that there is a general object *inflected-word* which stands model for all words that are inflected. Suppose, furthermore, that *inflected-word* contains an aspect to compute a pronunciation by concatenating the pronunciations of its prefix, stem and suffix:

```
(DEFOBJECT INFLECTED-WORD
  (PRONUNCIATION
    :FUNCTION (SELF)
    (CONCATENATE 'STRING
                 (PRONUNCIATION (PREFIX SELF))
                 (PRONUNCIATION (STEM SELF))
                 (PRONUNCIATION (SUFFIX SELF)))))
```

The keyword `:FUNCTION` denotes that the aspect defines a function to compute. A call to the generic function *pronunciation* will apply the function to the object. Suppose, e.g., that *weak-verb* is defined as follows:

```
(DEFOBJECT WEAK-VERB
  (PAST-PARTICIPLE-PREFIX
    (A MORPHEME
       (PRONUNCIATION "G@")))
  (PAST-PARTICIPLE-SUFFIX
    (A MORPHEME
       (PRONUNCIATION "+t")))
  (PAST-PARTICIPLE
    (A WEAK-VERB INFLECTED-WORD
       (PREFIX :IF-NEEDED #'PAST-PARTICIPLE-PREFIX)
       (SUFFIX :IF-NEEDED #'PAST-PARTICIPLE-SUFFIX)))
  ...)
```

The pronunciation of the past participle inherits from *inflected-word* and can be computed by a call to the generic function *pronunciation*[4]:

[4] For the reader who is interested in the procedure call mechanism we add the following note. The usual *lambda binding* in LISP is used when the function is applied to its arguments. With the appropriate bindings, the body of the function is executed and returns a value. Note that it is not required for any arguments except the first one to be an object. Only the first argument acts as a selector for the definition. A generic function must therefore take at least one argument. While in some object-oriented languages, `SELF` is a reserved word denoting the object itself, this is not the case in CommonORBIT. It is a normal parameter of the function and could have any other name.

```
(PRONUNCIATION (PAST-PARTICIPLE 'WERKEN))
"G@+wErk+t"
```

Note how in the past-participle of weak-verb, the aspects *prefix* and *suffix* dispatch to other aspects (e.g., the past participle dispatches *suffix* to *past-participle-suffix*). These aspects need to be computed only once for each object. The resulting value can then be stored and should not be recomputed on subsequent access. The aspect type `:IF-NEEDED` achieves precisely this goal by storing the value as a *memo*. An aspect of type `:IF-NEEDED` is called like one of type `:FUNCTION` but *memoizes* (or *caches*) the result of the computation by redefining the aspect as one of type `:VALUE`. This saves computational effort at the expense of some extra storage. Memoization is *lazy*, i.e., the value is not computed until it is needed, i.e., when the generic function is first called on the object.

6.4 Concluding remarks

The pervasive role of defaults in the lexicon suggests that default inheritance is almost indispensable in any theory of language. Defaults have been expressed in various ways in linguistic theories, e.g., as redundancy rules, blocking, paradigms, etc. Default reasoning also seems to play a role in several other cognitive processes, including common sense reasoning. This suggests that—in this respect—linguistic processing is not basically different from other cognitive processing. A grammar is rendered considerably simpler if default inheritance is factored out as a metatheoretical primitive of the cognitive system. A theory of language then fits within a general theory of symbolic models, and general-purpose knowledge representation language can be used to express linguistic concepts. An object-oriented grammar represents linguistic units such as phrases, words, morphemes, phonemes, and semantic structures uniformly as computational objects. Regularities and exceptions are represented in a specialization hierarchy which is governed by sophisticated inheritance mechanisms. The object-oriented language CommonORBIT provides such mechanisms.

In a process theory of language, the *software engineering* advantages of object-oriented programming must not be overlooked. The object-oriented paradigm is especially suited to simulation tasks (including natural language processing) because it manages the complexity of the underlying knowledge base. Object-oriented programming provides important processing advantages: locality of information, reduction of the search space, and easy propagation of changes.

However useful an object-oriented grammar may be, some empirical questions remain. In practice, object-oriented models are not as conceptually clear as a theoretical linguist would like. This is partly due to differences between object-oriented languages. Also, the fact that the object-oriented paradigm itself is flexible and powerful means that the number of possible organizations of the same domain is large. This number can perhaps be reduced by an *information analysis* of the domain, although this is hindered by the abstract nature of the concepts involved (Van der Linden et al., 1989). The representational power remains a problem from a psycholinguistic viewpoint, because

there is little empirical evidence for particular hierarchical organizations of knowledge. Therefore this chapter has drawn mainly on evidence from *linguistic* rather than psychological phenomena. Finally, there is a lack of dynamics. Hierarchies must be explicitly programmed; although in principle they are flexible, it is unclear how they originate and change in a learning task. In short, much empirical work remains to be done.

The present discussion has been limited to *symbolic* representations. Recent work has indicated that connectionist models are capable of learning hierarchical categories and can represent them at a *subsymbolic* level of representation. Elman (1989) reports an experiment where a *simple recurrent network* is given the task of predicting words in very short sentences. A hierarchical cluster analysis of the average hidden unit activation patterns reflects the discovery of several major categories of words, e.g., *noun* vs. *verb;* nouns are subdivided into the classes *animate* vs. *inanimate;* animates are subdivided into *humans* and *animals*, etc. The fact that a substantial hierarchy can clearly be distinguished is an important finding, but the prediction task has its limitations. Further experiments must reveal whether inheritance hierarchies such as the ones proposed in this chapter can be learned and used by a connectionist model.

7 Default inheritance in linguistic theory

A grammar in which *categories* are viewed as objects assigns an important place to the lexicon, because categories are essentially abstractions over a set of lexical items. It is therefore not surprising that during the development of a *lexicalist* view of grammar during the seventies, the need for a general mechanism to eliminate redundancy was clearly felt. This development is briefly reviewed. AI research developed more general and sophisticated mechanisms on the basis of default reasoning. It is shown how such general mechanisms were applied in early AI approaches to language processing, and were subsequently incorporated in new grammar formalisms.

7.1 A lexicalist framework

Traditional linguistics distinguishes a grammar consisting of a set of rules and a lexicon consisting of a set of words. When the lexicon is large, as is typically the case in natural languages, the question is, how to describe it to capture its regularities? The question can also be put as follows: how can the lexicon be reduced in favor of a more economic set of rules? One approach tried by transformationalists (cf. Chomsky, 1965) was to describe the relation between words by deriving whole sentences from others which contain different words, e.g., the transformation which derives (1b) from (1a).

(1) a. Bill decided to go.

b. Bill's decision to go.

The *lexicalist hypothesis*, started by Chomsky (1970), rejects such transformations, relating lexical entries by means of *lexical rules* instead. Chomsky's treatment of nominalization has later been extended by others to various word formation processes. Eventually the lexicalist movement resulted in a massive relocation of linguistic knowledge to the lexicon. From the very start of lexicalism, the concept of *lexical exceptions* was felt to be important: "transformations are supposed to handle the fully regular processes, not processes governed by lexical exceptions" (Hoekstra, Van der Hulst & Moortgat, 1980:4). There had long been rules that handle phonological regularities (Stanley, 1967) although little had been done to formalize them. These *redundancy rules* were rediscovered along with morphology in the wake of Chomsky's seminal work. Jackendoff (1975), e.g., recognizes the problem "to develop a notion of lexical redundancy rules which permits an adequate description of the partial relations and idiosyncracy characteristic of the lexicon."

The question was then put forward, what the explanatory role of lexical rules is. Since the lexicon is productive, rules dealing with word structure account for the formation of new words. But they also describe the internal structure of existing words. Jackendoff (1975) focused on the latter, he calls the *full-entry* theory of the lexicon. In this theory, redundancy rules do not play an active role in a derivation, but rather function as an evaluation measure for the lexicon. Thus, all derivations are stored in the lexicon. The fact that some lexical rules can be used to create new words is of secondary importance. In a similar vein, Halle (1973) proposes that the dictionary contains all words, including all fully inflected forms of the language, so that the speaker "needs to invoke the word formation component only when he hears an unfamiliar word or uses a word freely invented" (Halle, 1973:16).

The full-entry view is challenged by Siegel (1978), who stresses the dynamic function of lexical rules and proposes that all non-atomic words are generated each time anew. The resulting word can later be discarded, since another, equivalent word will be generated the next time it is needed. This view is sometimes called the *impoverished entry* theory of the lexicon. In contrast, the full-entry theory reflects precomputation and storage of forms; for each new word, the word formation rules operate only once.

Both views can be represented in an object-oriented programming language. In CommonORBIT (see Chapter 6), the impoverished-entry theory can be modeled by the use of `:FUNCTION` aspects, which perform the same computation every time they are invoked. The full-entry theory can be implemented by means of an `:IF-NEEDED` aspect in CommonORBIT. Once a word is computed, it is stored in memory and can be accessed without reinvoking the rules that generated it. It then also falls within the scope of the lexical redundancy rules and could give rise to more words.

Hetzron (1975) takes an intermediate position: observing that frequently used words have more exceptional morphology than average, he presumes that the speaker must use readily stored material only for those exceptional forms, while everywhere else the word formation component is invoked. Hetzron proposes the following mechanism:

> "Technically, this can be represented by a disjunctive set of rules where idiosyncratic or 'exceptional' formations are listed with as much explicitness as necessary, while the general word formation rules would apply afterward, with the power to apply 'to the rest'."
> (Hetzron, 1975)

Hetzron's *disjunction* is also known to linguists as *blocking*—when an irregular form exists, the simultaneous production of a regular one must be prevented. This disjunction is automatic in object-oriented languages, because it is in fact the essence of default inheritance: either an object has a definition of its own (the "exceptional formations"), or if it has none it inherits a definition from a more general object.

7.2 Inheritance in AI models of language

In the AI and Computational Linguistics literature, several independent lines of research converged on the application of frame-based and object-oriented languages to natural

language processing. Initially, such representations were used for semantic processing and representations of the domain of discourse, e.g., in the form of semantic networks, case structures, etc. (cf. Jacobs, 1987, for a more recent approach in this field). Later, representations based on frames and structured objects were also applied in the areas of syntax and morphology. Recently, the object-oriented paradigm is also gaining ground in phonology, orthography and lexicography through the efforts of Daelemans (1987a,b, 1988a,b, 1989). The following will be confined to some important developments with respect to an object-oriented representation of *syntactic* and *morphological* categories.

7.2.1 PSI-KLONE

Bobrow and Webber (1980) is one of the first accounts of an AI application where inheritance is consistently applied in a representation of linguistic knowledge. In the context of RUS, a system for natural language parsing and interpretation, Bobrow and Webber posit an independent declarative knowledge base for linguistic knowledge. The formalism for this representation is PSI-KLONE, a derivative of the object-oriented language KL-ONE. Inheritance is used to organize linguistic knowledge efficiently in terms of grammatical categories. Some of the basic inheritance relations are depicted as solid arrows in Figure 7.1.

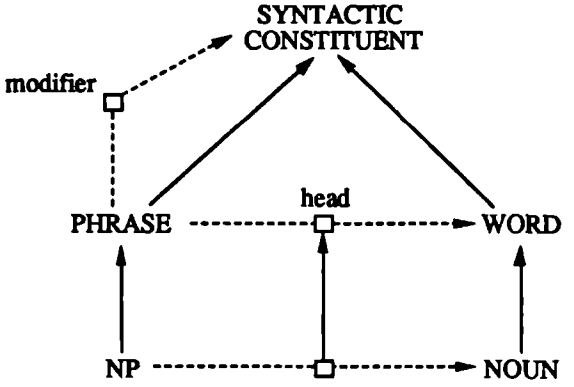


Figure 7.1
PSI-KLONE network

Syntactic relations in PSI-KLONE are expressed as slots in a KL-ONE object, shown here with dashed arrows. E.g., the head of a phrase is represented as a slot which is filled by the object *word*. *Structured inheritance*, indicated as arrows between small squares on the slot arrows, models a slot after one higher in the hierarchy. Structured inheritance has been described as the ability to "... preserve a complex set of relations between description parts as one moves down the specialization hierarchy" (Brachman & Schmolze, 1985:177). In other words, inheritance is not limited to simply sharing a value, but it models an object and the network of all its associated objects after one higher up in the hierarchy. This mechanism, which is central in KL-ONE, is also present in CommonORBIT (see also Section 8.3.3).

In RUS, this object-oriented representation is used by a process called Incremental Description Refinement, which first determines which descriptions are compatible with an object known to have a given set of properties, and then refines this set of descriptions as more properties become known. The domain of structured objects is therefore complemented by a space of descriptions and rules that determine which descriptions are applicable.

7.2.2 Conceptual Grammar

A similar view of language processing as refinement of descriptions is proposed in the work of Steels and De Smedt. Early work by Steels (1978b,c) introduced the theory of Conceptual Grammar, a representation of linguistic knowledge based on *frames* (Steels, 1978a, 1979). A Conceptual Grammar presupposes not only a single representation of linguistic knowledge for a variety of language processing tasks, but also a single type of representational device for all components of the grammar. Because a language is seen as a network of concepts (or *frames*), the acquisition of language can be viewed as a concept formation process. Thus all psychological findings about concept formation become relevant for language learning.

In contrast to other formalisms like ATNs and TGG, a Conceptual Grammar is completely declarative and does not use procedural notions like *affix-hopping, deletion, move-α, set register,* etc. The question, how linguistic knowledge is used, is left to a general inference mechanism, which means that the grammar itself is simplified considerably. It is thus neutral with respect to language generation or language analysis. In language generation, the cognitive system provides a description of the goals, beliefs, etc. of a certain communication situation. This activates certain frames which fit the description. By means of the propagation of constraints among frames, a coherent surface description of linguistic forms is created.

Steels and De Smedt (1983) further developed this idea, partly influenced by Kay (1979). They propose a frame-based representation where linguistic structures are viewed as collections of *descriptions*. A description is based on a concept, i.e., a category. *'Has number plural', 'is infinitive', 'is subject of sentence', 'fulfills the agent role'* are examples of descriptions. Each description gives partial information about one of the linguistic objects which are constructed to establish a communication act. A grammar is then seen as a large number of descriptions grouped in structures called *frames* which are organized in an inheritance hierarchy. The hierarchy depicted in Figure 7.2 represents the specialization relations between Dutch articles:

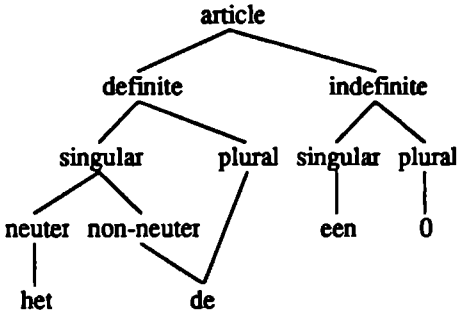


Figure 7.2
Hierarchy of Dutch articles

All relations in this hierarchy are XOR links, and therefore the multiple generalization link above *de* is fundamentally different from ordinary multiple inheritance, which is an AND link. Thus the hierarchy is more like a *decision tree*. It is represented in a frame-based representation language by stating in each frame which frames are more general, as well as how this frame can be refined into more specific frames. E.g., the frames for *article* and *definite article* are defined as follows:

```
(FRAME ARTICLE
   (REFINEMENT
     (WHEN ((WITH DEFINITENESS DEFINITE) DEFINITE-ARTICLE)
           ((WITH DEFINITENESS INDEFINITE) INDEFINITE-ARTICLE))))

(FRAME DEFINITE-ARTICLE
   ARTICLE
   (WITH DEFINITENESS DEFINITE)
   (REFINEMENT
     (WHEN ((WITH NUMBER SINGULAR) SINGULAR-DEFINITE-ARTICLE)
           ((WITH NUMBER PLURAL) PLURAL-DEFINITE-ARTICLE))))
```

Language processing can then be viewed as the construction of descriptions which are appropriate within a given communication act, be it parsing, production, or other linguistic activity). The knowledge in these frames is specified in two distinct ways, one which allows a generalization to more general frames, and the other which allows refinement to more specialized frames. When a problem needs to be solved, descriptions are generalized or refined (or both) until a description is obtained that contains the required knowledge[1]. Suppose that in a production task, the following descriptions are given and the spelling needs to be found:

```
ARTICLE
(WITH DEFINITENESS DEFINITE)
(WITH NUMBER SINGULAR)
```

[1] This can be likened to forward chaining and backward chaining in rule-driven systems. It is significant that both directions may need to be explored regardless of the task.

Alternative descriptions will be constructed until a description providing the spelling has been found. Put briefly, this process consists of refining the frame *article* until the frame *het* has been found. The hierarchy is thus used as a decision tree, not unlike the choice systems in *systemic grammar*. Inversely, when a spelling is given and a number or category needs to be found, generalizations are tried until one is found which contains the necessary information. When a frame has multiple generalizations, alternative descriptions will result. These can be further reduced when more information becomes known, e.g., when an article is combined with a noun in an NP. Agreement relations are expressed by variable bindings, e.g., in the following definition for NP, the definiteness of the article is the same as that of the NP:

```
(FRAME NOUN-PHRASE :NP
  (WITH HEAD NOUN
        (WITH NUMBER (NUMBER :NP))
        (WITH MEANING (MEANING :NP)))
  (WITH DETERMINER ARTICLE
        (WITH DEFINITENESS (DEFINITENESS :NP))
        (WITH GENDER (GENDER (HEAD :NP)))
        (WITH NUMBER (NUMBER :NP)))
  (WITH UTTERANCE SEQUENCE
        (WITH FIRST (SPELLING (DETERMINER :NP)))
        (WITH SECOND (SPELLING (HEAD :NP)))))
```

The problem solving process in either parsing or generation will consist of developing alternative descriptions for entities such as an NP while keeping internal consistency by means of the variables. This approach is somewhat similar to that of a Definite Clause Grammar (e.g. Schotel, 1987), which uses feature variables as extra arguments in *terms* in order to enforce agreement. The activation of frames to develop alternative descriptions is a parallel distributed process, in which several structural hypotheses may be developed simultaneously.

7.3 Inheritance in recent grammar formalisms

Whereas mainstream linguistics in the past decades has been oriented toward the finding of 'cute facts' and providing a 'cute analysis' for them, there is now a new breed of researches which draws from psychology and computer science as well as from linguistics and is concerned more with realistic linguistic formalisms (Gazdar, 1987). Several new grammar formalisms which were developed during the eighties gave more substance to both a lexical view of grammar and the use of default inheritance in the linguistic domain. I will outline *Word Grammar* and two variants of *Unification Grammar*. Despite substantial differences in linguistic representation, these formalisms are all clearly committed to inheritance relations between linguistic categories, and hence can rightfully be called object-oriented grammars.Unlike the early AI experiments, which were aimed primarily at the implementation of practical systems, these new grammars are aimed at the formal definition of new theories of language.

7.3.1 PATR-II templates

Unification-based grammars (Shieber, 1986) are lexically oriented in the sense that they have a simple combination mechanism—*unification*—which operates on quite complex lexical items. Lexical items are represented as *feature structures* which are essentially functions mapping features onto values (see Section 3.1). Lexical entries often share much common structure in the form of common features. E.g.[2], all verbs in a unification grammar may share feature structure (2), while all intransitive verbs may also share the additional syntactic subcategorization information in (3):

(2) [cat = V]

(3) $\left[\text{subcat} = \begin{bmatrix}\text{first} = [\text{cat} = \text{NP}] \\ \text{rest} = \text{end}\end{bmatrix}\right]$

Due to the rich structure of subcategorization information, the encoding of generalizations in such a formalism is important. In PATR-II, *lexical templates* are feature structures which are labeled so that they can be used in other lexical entries. E.g., feature structure (2) can be assigned the label *Verb* in the following way:

```
Let Verb be <cat> = V.
```

Similarly, (4) can be labeled with *Intransitive*; at the same time, it can be specified that this feature structure always includes *Verb*:

```
Let Intransitive be Verb
     <subcat first cat> = NP
     <subcat rest> = end
```

Since templates may include other ones, this leads to a hierarchical organization where feature structures are shared. This structure sharing between templates is essentially the same as inheritance in object-oriented languages. Multiple inheritance is allowed to combine features from assorted structures in new categories, e.g.:

```
Let ThirdSing be ThirdPerson Singular.
```

Lexical items can likewise be composed of several categories:

```
sleeps ⟼ Intransitive ThirdSing
     <head trans pred> = sleep.
```

However, unification disallows conflicts between local and inherited information. The PATR-II system provides a special method of combining structures by *overwriting* feature values in addition to normal unification. By allowing feature structures lower in the hierarchy to overwrite information higher up in the hierarchy, the effect of *default*

---

[2] The examples in this subsection are simplified from Shieber (1986:55ff). We will not be concerned with the actual content of the feature structures.

inheritance can be achieved. Such overwriting assignments have to be explicitly stated by means of a double arrow in the template definition, e.g.:

```
Let Finite be Verb
     <head agreement case> => nominative.
```

However, such overwriting has a drawback compared to a more complete default reasoning system: since overwriting is destructive and the PATR-II system does not use dynamic inheritance, overwriting devastates the order independence of the system. CommonORBIT is much more order independent in this respect.

7.3.2 Head-driven Phrase Structure Grammar

In the context of Head-driven Phrase Structure Grammar (HPSG), structure sharing in lexical representation—using default inheritance—is proposed by Flickinger, Pollard and Wasow (1985) and by Sag & Pollard (1987). HPSG is a derivative of GPSG (Gazdar, Klein, Pullum & Sag, 1985) in which much of the linguistic knowledge is relocated from the syntax to a richly structured lexicon. Lexical rules capture both inflectional and derivational regularities among lexical entries. Inheritance is seen mainly as a way to simplify the lexicon and thus make possible an efficient, readily modifiable, large natural language system.

Flickinger, Pollard and Wasow (1985) use inheritance in much the same way as De Smedt (1984), although they are not primarily concerned with morphological subcategorization, but rather with syntactic subcategorization. They propose a lexical organization where subcategorization frames are inherited. For this, they do not rely on explicit *overwriting*, as in PATR-II templates, but they use a full-fledged inheritance mechanism based on a class/instance distinction. E.g., a class *verb* is defined which includes subclasses *base* and *finite*. Having specified that the feature *major* for the class *verb* is V, it not necessary to stipulate this again for the subclasses *base* and *finite*, since the feature will be inherited by each subclass. Inheritance is transitive: each instance of the class *finite*, e.g. the word form *works*, will inherit the value *finite* for the feature *form* directly from the class *finite* and will indirectly inherit the value V for *major* from the class *verb*. Two distinct modes of inheritance are deemed necessary by Flickinger, Pollard and Wasow: in addition to a *normal* mode, which retrieves the most specific value, there is a *complete* mode, which collects all values for a given slot from the classes and superclasses of a given frame. However, it seems that the complete mode of inheritance is necessary only because of an artificial grouping of syntactic features in one slot called *features*. If each feature were accessed separately, then the normal mode of inheritance would suffice.

Multiple inheritance is used to inherit information from more than one class. This possibility is applied, e.g., to assign both a syntactic and a morphological class to a lexical entry: the intransitive finite verb *works* is not only an instance of the class *finite*, but also of *intransitive* (presumably also a subclass of *verb*), from which it inherits valence information. This representation, which is depicted in Figure 7.3, is an interesting use of multiple inheritance as a way of composing knowledge from two very different sources.

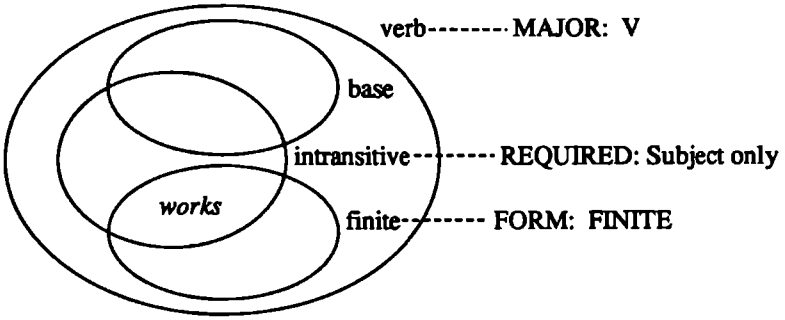


Figure 7.3
Multiple inheritance in HPSG using classes

The knowledge in each lexical entry is thus composed out of a morphological object, which represents its morphological behavior, as well as a syntactic object, which represents its syntactic subcategorization. However, it can be questioned whether semantico-syntactic information, such as subcategorization, syntactic features, etc., should be associated with words. In Section 4.3 it is argued that it is preferable to assign this information with *phrases* instead of words.

The mechanisms for inheritance proposed by Flickinger, Pollard and Sag (1985) can account for both the full-entry theory or the impoverished-entry theory of the lexicon, since they can be applied without modification either before a word is ever used, or only when needed. The cost of storing a redundantly specified lexical item against computing it anew each time is only dependent on space/time trade-offs in their system. Thus inheritance serves the economy of the system as a whole.

7.3.3 Word Grammar

Hudson's (1984) Word Grammar is by definition an object-oriented grammar, for he views language explicitly as "a network of entities related by propositions". One of the fundamental relations between these entities, the *model* relation, corresponds to default inheritance. This is the relation between a *model* and an *instance*, e.g., between *in* and *preposition* and between *preposition* and *word*, but also between a particular token of *in* on a particular occasion and the lexical entry *in* in the dictionary. The model/instance distinction thus covers a class/member distinction as well as a type/token distinction. Instances inherit properties from their models by means of an inheritance mechanism, which is seen as a universal cognitive principle which happens to be especially useful in linguistics. Hudson defines this *Selective Inheritance Principle* as follows:

> "If I is an instance of M, then any proposition which applies to M must apply to I as well (with 'I' substituted in it for 'M'), provided this proposition does not contradict any proposition which is already known to apply to I." (Hudson 1984:18)

Clearly, this is yet another formulation of default inheritance as it is used in the object-oriented paradigm. Hudson apparently does not view his model/instance distinction as

corresponding to a strict class/instance distinction, which is postulated in many object-oriented languages, e.g., CLOS and Flavors. It seems that specific words can themselves stand model for other words, as exemplified in Figure 7.4. Thus, the model acts like a stereotype, and model/instance relation corresponds more to a prototype relation much like that in CommonORBIT.

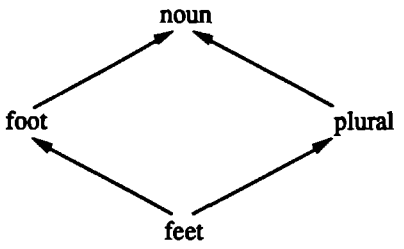


Figure 7.4
Multiple inheritance in Word Grammar using a model/instance relation

Word Grammar exploits this possibility by allowing the output word of lexical rules to be instances of the input word, e.g., the word *adjective* is a model for the derivation *non-adjective*. This is generally not the case in HPSG, where information is instead copied from the input frame to the output frame of a lexical rule. Thus the HPSG approach suffers from a certain redundancy.

As a motivation for inheritance as a general cognitive mechanism, Hudson proposes the general psychological principle of minimum storage. The question, to what extent predictable information is stored in people's heads, and to what extent it is not, is ultimately an empirical one, and is not likely to be answered soon. It is very well possible that a grammar with maximum generalization is predicting less stored information than people actually have in their heads. But even with this psychological uncertainty, it is useful to explore the limits of generalization. Hudson formulates maximum generalization as the linguist's *economy principle*:

> "Never record any property more than once in relation to any entity" (1984:29).

Principles of economy are the ultimate motivation of generalization and the formation of categories. Lakoff (1987) proposes a similar notion of *cognitive ecology*. In his theory of cognitive linguistics, and indeed cognition in general, *prototype* theory is used for categorization. There are central and non-central categories. Less central categories are characterized by those minimal differences that distinguish them from the more central categories they are based on. Any property that can be predicted from more central categories is redundant. Lakoff defines *relative motivation* for each category as follows:

> "The more the properties of a given category are redundant, the more it is motivated by its ecological location, and the better it fits into the system as a whole." (Lakoff, 1987:493)

The relative motivation of all categories combined makes up the motivation of the system as a whole: an optimal structure for a linguistic system is one that maximizes total *systemic redundancy* in the grammar and the cognitive system taken together.

7.4 Concluding remarks

In a sense, lexicalism can be seen as a return to traditional European grammar. These grammars were organized in terms of linguistic categories and describe the relations between those categories in a rather declarative way. Typically, these grammars have a chapter on the *noun*, one on the *verb*, etc. But whereas traditional grammars sufficed with an informal and intuitive account of regularities and exceptions, there is now an awareness of the need for more formally specified linguistic representation. Linguists have recognized and formulated regularities in the lexicon as specific grammar rules such as *redundancy rules*, *blocking*, etc. Using inheritance as a general representational mechanism, these rules become less *ad hoc* but follow from a cognitive architecture based on hierarchical default reasoning.

In this chapter I have mainly focused on the application of general purpose inheritance mechanisms to linguistic representation. It must be mentioned that some inheritance formalisms have been developed specifically for linguistic purposes. The PATR-II system discussed above is an example of such a system. Another new development is DATR, a formalism proposed by Gazdar and Evans (1989) specifically for the representation of lexical entries. The default mechanism in DATR is extended to cover *paths* of nodes: each property of a node by default also holds for all extensions to the path at that node. E.g., if the *past* of a *verb* denotes a certain value, then the path *past tense singular third* by default also has that value. Such default inheritance in paths seems to be useful in the representation of lexical entries, but it remains to be seen if such representation is felicitous in all paths, not just in the given example.

A remaining question is, in how far an object-oriented representation can be declarative. I have been mostly concerned with the structure of the specialization hierarchy, and not so much with the question of how the knowledge associated with each object is actually expressed. In a straightforward object-oriented approach, this knowledge could be expressed, e.g., as generic functions. Even if some of these generic functions contain declarative knowledge, other functions must eventually perform explicit computations to perform string concatenation, etc. Calder (1989), however, proposes a notation for morphological relationships and lexical rules based on *string unification*. A string specification in this system is a sequence of characters from an alphabet and possibly also containing variables which stand for an arbitrary character. Strings can be unified by matching corresponding elements; where variables occur, they are bound to corresponding characters. Calder further defines lexical entries in which strings are associated with a set of grammatical properties which are intended to be similar to PATR-II templates. In a *paradigm*, lexical items are related by lexical rules. String forms and associated lexical rules can be inherited from a more general paradigm. Summing up, it seems to be possible to express arbitrary operations on

strings in a declarative way using string unification rather than procedural string manipulations.

8 An object-oriented implementation of Segment Grammar

Segment Grammar (SG; see Chapter 3) is implemented in an object-oriented framework by uniformly representing all important grammar units—syntactic segments, syntactic categories (phrases, words) and syntactic features—as structured objects. By way of example, all the framed entities in the example of Figure 8.1 are represented as CommonORBIT objects.

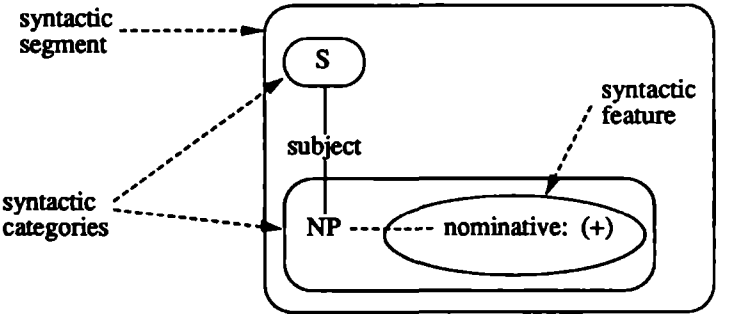


Figure 8.1
Syntactic segments, categories and features as CommonORBIT objects

This uniform representation makes allows the use of general mechanisms for the application and manipulation of linguistic knowledge. In particular, SG profits from the inheritance mechanism in CommonORBIT which promotes generalization and avoids redundancy. It will be shown how inheritance is applied in SG at all levels of structure.

8.1 Feature structures as objects

Unification-based grammar can be joined to object-oriented representation by representing feature structures as *objects*. The *aspects* (slots) of an object are then interpreted as features. E.g., the following CommonORBIT object definition is equivalent to feature structure (1):

```
(A FEATURE-OBJECT
   (CATEGORY 'NP)
   (PLURAL '-)
   (NOMINATIVE '+))
```

(1)
$$\begin{bmatrix} \text{category} = & \text{NP} \\ \text{plural} = & - \\ \text{nominative} = & + \end{bmatrix}$$

However, the value of a feature need not be atomic. In order to facilitate feature sharing, features can themselves be represented as objects. Thus they can also be unified (see Section 8.3.2). Features are the simplest objects, with just two aspects, one for the name, and one for the value. Binary features are features with just two possible values.

```
(DEFOBJECT SYNTACTIC-FEATURE
  FEATURE-OBJECT
  (NAME)
  (VALUE))

(DEFOBJECT BINARY-FEATURE
  FEATURE
  (VALUE '(+ -))

(DEFOBJECT PLURAL-FEATURE
  BINARY-FEATURE
  (NAME 'PLURAL))
```

Using such features, feature structure (1) can be redefined as:

```
(A FEATURE-OBJECT
   (CATEGORY 'NP)
   (PLURAL (A PLURAL-FEATURE (VALUE '-)))
   (NOMINATIVE (A NOMINATIVE-FEATURE (VALUE '+))))
```

Now such a definition is more verbose, but fortunately we can use inheritance to define features structures as combinations of other ones. 'Mixins' may be defined to hold more specific feature values, e.g., an object *singular* which contains only the value '-' for the feature *plural*:

```
(DEFOBJECT SINGULAR
  (PLURAL (A PLURAL-FEATURE
             (VALUE '-))))
```

Using such mixins as abbreviations, the CommonORBIT definition of a feature structure can be much more concise. The following definition again describes feature structure (1). Figure 8.2 shows part of the hierarchy which is relevant to this definition.

```
(A SINGULAR NOMINATIVE NP)
```

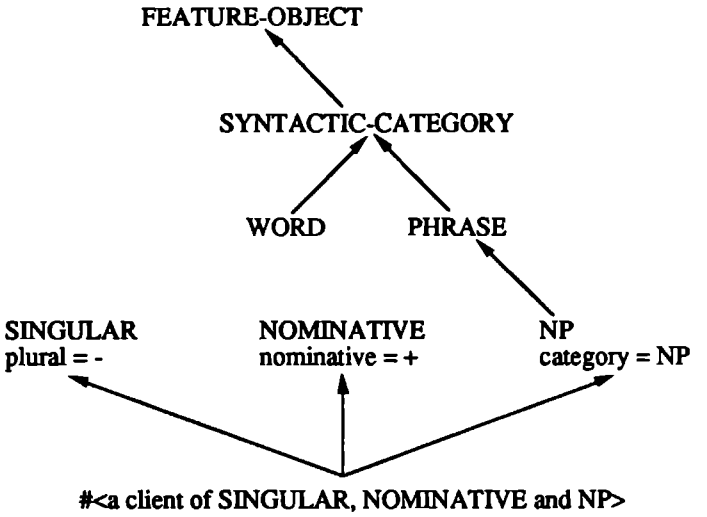


Figure 8.2
Example hierarchy of objects as feature structures

I must add a note here about the functional nature of feature structures. Feature structures are usually seen as functions which map features onto values; in fact, they are sometimes called *functional* structures for this reason. In a sense, an object-oriented representation in CommonORBIT does the reverse in the sense that features are *generic* functions which map feature structures onto values. If they were normal functions, this might be somewhat problematic, for those functions would have an infinite domain. However, since the definition of a generic function is *distributed* among feature structures, function and argument can in theory be reversed, and thus the problem is avoided. Some object-oriented languages (e.g. FLAVORS: Weinreb & Moon, 1980) do actually implement objects as functions.

8.2 Unifying CommonORBIT objects

When feature structures are represented as objects, then how is their unification defined? CommonORBIT provides a unification operation, called `UNIFY`, which unifies two objects by merging all the information contained in both objects into one. Not all aspects of an object are always essential for unification. The aspects involved in the unification of an object are therefore obtained by the generic function `FEATURES`. The `UNIFY` operation succeeds if the values of all these features in both objects match in one of the following ways:

1 If the values are both objects, then the objects are unified. Match if this unification succeeds; the unification becomes the new feature value. Thus, unification is recursive[1].

2 Otherwise, the values are interpreted as sets (by coercing to lists) and the intersection is computed. However, the value UNDEFINED matches anything. Match if the result of the intersection is non-NIL; the intersection becomes the new feature value.

If unification succeeds, then the two objects are merged into one and the new feature values are stored into this one object, as well as all other information which was present in both objects. If unification fails, the result is UNDEFINED.

Although unification is applicable to all CommonORBIT objects, there is a special prototype FEATURE-OBJECT with convenient defaults for feature-structures. This prototype can be directly or indirectly used as a prototype by all feature structures (e.g. in Figure 8.2). For SG, there is also a prototypical *syntactic-category* with convenient defaults. This object states, among other things, that the feature *category* is a required feature for unification. This prevents the unification of two different categories, e.g., S and PP.

A general problem with the merging of objects is, what happens with existing references to objects when they are merged. A practical solution which is adopted by CommonORBIT is to implement an object as a two-level structure. The first level, which is called the *reference*, contains a pointer to the second level, which is called the *referent* of the object. All information about an object (except the name of a named object) is stored in its referent. Normally, each reference has its own referent which denotes an individual external entity. We can merge two existing objects by transferring all information from one referent to the other and subsequently redirecting the pointer from the 'emptied' referent to the 'filled' one. The two objects then share a referent, which remains accessible via both references. A schematic example is given in Figure 8.3.

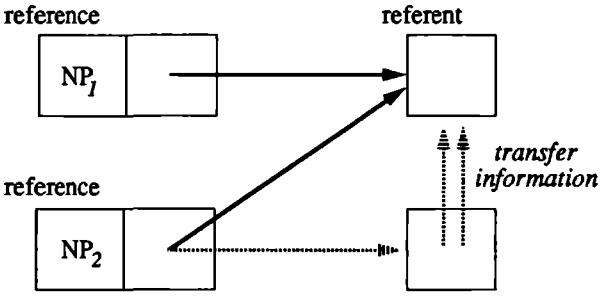


Figure 8.3
Merging objects during unification

[1] The full recursive power of unification in CommonORBIT is not strictly necessary in SG (see Section 3.1).

Decoupling the representation of an object and its referent raises the question, how equality of objects must be defined. Two distinct equality tests are provided. The predicate OQL tests whether two CommonORBIT objects are denotationally equal, i.e., whether they share the same referent. The normal EQ test in Common LISP can be used to test whether two objects are identical references.[2]

8.3 Syntactic segments as objects

Objects representing syntactic segments have aspects for the root, foot, and arc label, and also for the features which are shared between root and foot ('agreement features'). The segment NP-head-NOUN could be defined as follows:

```
(DEFOBJECT NP-HEAD-NOUN
  SYNTACTIC-SEGMENT
  (ROOT (A NP))
  (ARC 'HEAD)
  (FOOT (A NOUN
           (POSITIONS '(5))))   ;word order possibilities
  (AGREEMENT-FEATURES '(PERSON PLURAL GENDER NOMINATIVE))
  )
```

8.3.1 Knowledge sharing between segments

Segments may inherit from other segments. E.g., a lexical segment with the Dutch noun *watermeloen* (watermelon) delegates to the prototypical NP-head-NOUN segment. The segment contains syntactic knowledge, while its foot contains morphological knowledge (see Section 4.3).

```
(DEFOBJECT WATERMELOEN-SEGMENT  ;a lexical segment
  COUNTABLE NP-HEAD-NOUN
  (FOOT (A WATERMELOEN-NOUN)))

(DEFOBJECT WATERMELOEN-NOUN     ;a word
  COMPOUND-NOUN
  (COMPONENT-S (LIST (A WATER-NOUN)
                     (A MELOEN-NOUN))))
```

Segments can be organized efficiently in a specialization hierarchy. E.g., the segments NP-head-NOUN and NP-head-PRONOUN share much of their knowledge. This common knowledge can be stored in a more general segment, while the definition for NP-head-NOUN becomes more compact:

---

[2] Also, by creating a second reference pointing to an existing referent, we can create two *coreferential* objects. Due to the two-level structure, any subsequent changes to one object will also affect the other. Using coreference in object-oriented representation is further developed by Ferber and Volle (1988) who have implemented a logic for coreferential reasoning. Their system also allows references to be accessed from their referents, which is not possible in CommonORBIT.

```
(DEFOBJECT NP-HEAD-*
  SYNTACTIC-SEGMENT
  (ROOT (A NP))
  (ARC 'HEAD)
  (AGREEMENT-FEATURES '(PERSON PLURAL GENDER NOMINATIVE))
  ((FOOT POSITIONS) '(6))
  )

(DEFOBJECT NP-HEAD-NOUN
  NP-HEAD-*
  (FOOT (A NOUN))
  )
```

The specialization hierarchy of segments may exploit *multiple* inheritance. E.g., knowledge common to both segments S-subject-S and S-subject-NP is stored in a general segment S-subject-*. Knowledge common to all subordinate clauses is stored in *-*-S. This is schematically represented in Figure 8.4:

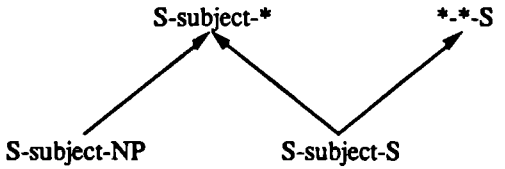


Figure 8.4
Some specialization relations between syntactic segments

8.3.2 Feature sharing

Feature sharing can be modeled in various ways. It can be based on feature unification: upon the creation of a segment, the features to be shared in the foot and the root are unified. Thus, the definition of a segment contains a specification of which features must be unified. This is the representation which is currently used in the CommonORBIT implementation of SG. But an alternative implementation is also possible. Rather than creating two features and unifying them, we could make just one feature and then assign it to the foot as well as the root. This can be achieved by referring twice to the same variable in Common Lisp[3]:

[3] A similar use of variables is found in the grammar proposed by Steels and De Smedt (1983; see also Section 7.2.2).

```
(LET ((P (A PERSON-FEATURE))
      (PL (A PLURAL-FEATURE))
      (G (A GENDER-FEATURE))
      (NOM (A NOMINATIVE-FEATURE)))
  (DEFOBJECT NP-HEAD-NOUN
    SYNTACTIC-SEGMENT       ;delegate to prototypical segment
    (ROOT (A NP
             (PERSON P)
             (PLURAL PL)
             (GENDER G)
             (NOMINATIVE NOM)))
    (ARC 'HEAD)
    (FOOT (A NOUN
             (PERSON P)
             (PLURAL PL)
             (GENDER G)
             (NOMINATIVE NOM)))
    ((FOOT POSITIONS) '(6))        ;word order possibilities
    )
```

8.3.3 Structured inheritance

Structured inheritance models a segment *as a structure* after one higher up in the hierarchy (see also Section 7.2.1). As already mentioned, syntactic categories on the root and foot of a segment are also defined as CommonORBIT objects. The structured inheritance mechanism in CommonORBIT establishes delegation relations between the root of a segment and that in its prototype, and likewise between the foot of a segment and that of its prototype. Suppose that the lexical segment for *watermeloen* inherits from NP-head-N, then the structured inheritance relations which are automatically established are depicted in Figure 8.5. From this hierarchy, it can be seen that WATERMELOEN inherits from the N at the root of the NP-head-N segment and can thus, e.g., obtain a value for the aspect POSITIONS by delegation.

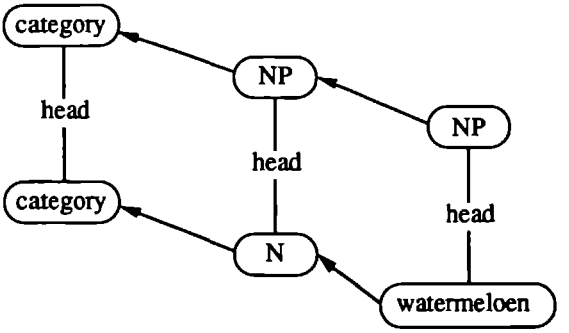


Figure 8.5
Structured inheritance in segments: the arrows represent inheritance relations which are automatically established[4]

Figure 8.5 also shows that the grammar and the lexicon are in a continuum: lexical segments are merely the most specific objects in the hierarchy of segments, and the phrases and words at their roots resp. feet are the most specific objects in the hierarchy of syntactic categories.

8.3.4 Implementing the 'destination' mechanism

In this section it will be shown how the destination mechanism (see Sections 4.4 and 5.3) is implemented in CommonORBIT. Each syntactic category should have a destination, which is determined by the ADDRESS of its mother node in the f-structure.

```
(DEFOBJECT SYNTACTIC-CATEGORY
  (DESTINATION
    :IF-NEEDED (SELF)
    (ADDRESS (TREE-MOTHER SELF)))
  ...)
```

Since words are always terminal elements in the c-structure, they are never mother nodes and hence never function as destinations. In other words, only phrases need to determine an address. By default, the address is the node itself:

```
(DEFOBJECT PHRASE
  SYNTACTIC-CATEGORY
  (ADDRESS
    :IF-NEEDED #'IDENTITY)
  ...)
```

Some S nodes are characterized by the fact that they do not function as destinations. Instead, they pass the responsibility for being a destination address to their mother in the f-structure. Moreover, such nodes are always NON-FINITE:

---

[4] For readability, the nodes are labeled with their categories. However, it must be kept in mind that the nodes themselves are instances of these categories.

```
(DEFOBJECT S-RAISED
  NON-FINITE S
  (ADDRESS
    :IF-NEEDED (SELF)
    ;; pass upward
    (ADDRESS (TREE-MOTHER SELF))))
```

Non-finite complement clauses are such nodes. S-RAISED nodes occur as the foot of a syntactic segment which expresses the COMPLEMENT grammatical relation:

```
(DEFOBJECT S-COMPLEMENT-S-RAISED
  S-*-*
  (ARC 'COMPLEMENT)
  (FOOT (A S-RAISED))
  ;; no positions because the foot has no destination
  )
```

Since the aspect ADDRESS in S-RAISED is of type :IF-NEEDED, the address must be computed only once. After that, the address is not found step by step, but is immediately accessible, no matter how deep the clause union construction is nested.

8.3.5 The passive rule

A final example concerns operations on lemmas. Lexical rules operate by deriving new lexical entries from other, existing entries. These rules are associated with lemmas (the phrasal part of lexical entries; see Section 4.3). E.g., the rule creating a passive counterpart of an active sentence lemma is in essence quite similar Bresnan's (1982) passive rule:

(2)

SUBJ	→	OBL by / 0
OBJ	→	SUBJ
0	→	Passive Participle

The following function transforms the case frame of an active lemma so that *subject* is changed to *by-phrase* and *direct object* to *subject*:

```
(DEFUN PASSIVIZE-CASE-FRAME (A-LIST)
  "Transform the a-list into another one where subject becomes
   by-phrase and direct object becomes subject."
  (LET ((NEW-A-LIST NIL))
    (DOLIST (L A-LIST NEW-A-LIST)
      (PUSH
        (LIST (FIRST L)
              (CASE (REST L)
                (SUBJECT 'BY-PHRASE)
                (DIRECT-OBJECT 'SUBJECT)
                (OTHERWISE (REST L))))
        NEW-A-LIST))))
```

The lexical rule is associated with the most general lemma that it is applicable to, i.e., the (active) S-LEMMA:

```
(DEFOBJECT S-LEMMA
  (PASSIVIZE                ;make a new passive lemma
    :IF-NEEDED (SELF)
    (LET ((NEW-LEMMA (A PASSIVE-LEMMA)))
      (IS-CLIENT NEW-LEMMA SELF)
      (SETASPECT 'CASE-FRAME NEW-LEMMA
                 (PASSIVIZE-CASE-FRAME (CASE-FRAME SELF)))
      NEW-LEMMA))
```

PASSIVE creates a new lemma which inherits from PASSIVE-LEMMA as well as from the original active lemma. In addition, the new lemma has a passivized case frame. The prototype PASSIVE-LEMMA for Dutch is defined so that it has the feature PASSIVE, has the auxiliary ZIJN in the perfect, and cannot be further passivized:

```
(DEFOBJECT PASSIVE-LEMMA
  PASSIVE PERF-ZIJN-LEMMA
  ;; since it is already passive, it cannot be passivized further
  (PASSIVIZE :VALUE UNDEFINED)
```

8.4 Concluding remarks

This chapter has described how an SG can be implemented in an object-oriented framework. By maximally exploiting the representational possibilities of the object-oriented paradigm, some of the workload in defining SG is shifted from the grammar formalism itself to the underlying representation. Thus, the definition of an SG can become much simpler while the representational primitives of the object-oriented representation can also serve other aspects of the linguistic system and of the cognitive system in general.

9 Parallelism in grammatical encoding

IPF is a computer model which allows the grammatical encoding stage in sentence generation to be distributed among a number of parallel processes. Each conceptual fragment which is input to the Formulator gives rise to a new process, which attempts to formulate only that fragment and then exits. The task of each formulator process consists basically of instantiating one or more segments and attaching these to the present syntactic structure by means of unification. A shared memory provides the necessary (and only) interaction between formulator processes and allows the integration of segments created by different processes into one syntactic structure.

In this chapter, the potential of parallelism in sentence generation will be investigated. It is shown how unifications in SG can be executed in parallel by using concurrent programming techniques.

9.1 Parallelism in language processing

Several arguments can be given to support the assumption that human language generation exploits parallel processing. A first argument relates to human information processing in general. Experiments on the division of attention (Allport, Antonis & Reynolds, 1972) has led to the 'multi-channel' hypothesis, which suggests "a number of independent, special purpose computers (processors and stores) operating in parallel and, at least in some cases, capable of accepting only one message or 'chunk' of information for processing at a time." It is further suggested that in general "any complex task will depend on the operation of independent, specialized processors, many of which are common to other tasks." Arguments for viewing processing components in language generation as such autonomous components are given by Levelt (1989).

A second argument for parallelism can be found in the psycholinguistic analysis of speech errors. Some kinds of fusion and omission errors seem to originate in the fact that the speaker processes more linguistic expressions at the same time. Due to either internal or external factors, the human language processing system may substitute unintended elements for the expected ones. Garrett (1975, 1980) assumes that computational simultaneity is a general condition for such interchanges.

A third argument is a neurolinguistic one. Circuit switching in the human brain is relatively slow compared to that in present-day computers. In order to account for the speed of human language processing—and indeed human information processing in general—an upper bound of about 100 single consecutive computational steps must be

placed on the processing of a sentence. This is vastly below the number of steps required by algorithms based on any present-day non-parallel theory of language.

There are several approaches to parallelism in language processing, and to cognitive processing in general. The first approach consists of the introduction of parallelism in a symbolic programming paradigm. The symbolic approach assumes an explicit representation of rules and concepts. Such a mentalistic view puts the grammar at the origin of linguistic behavior (cf. Chomsky, 1959). The second approach is to relinquish symbolic representations and instead represent knowledge at a subsymbolic level. Connectionist models do without an explicit, discrete representation of grammar rules and concepts. The grammar is then no more than an abstraction *a posteriori*. Regularities, and indeed mentality in general, are then statistically emergent properties of subsymbolic behavior (cf. Elman, 1989). In the present work, a symbolic approach to parallelism is explored.

9.2 Distributed grammatical encoding

Concurrent programming is the name given to programming notations and techniques for expressing (potential) parallelism and for solving the resulting synchronization and communication problems. Ben-Ari (1982) gives an introduction in concurrent programming as an abstract setting in which to study parallelism. These notations and techniques are essentially independent of whether the program is actually executed on a machine with multiple processors or is simulated as a time-sharing system on a single processor machine. IPF, the incremental parallel formulator, is currently implemented as a pseudo-parallel multi-processing system. In this section it is briefly shown how concurrent programming techniques can serve a parallel Formulator.

The control structures employed by IPF are implemented as high-level programming constructs written as an extension of Symbolics Common Lisp (SCL). SCL is a software environment with primitive operations for concurrent programming, e.g., for handling coroutines, atomic conditional assignment, and waiting. There is a process *scheduler* which handles multiprocessing. Some concurrent programming constructs will be explained now, and at the same time it will be indicated how they are applied in IPF.

A basic operation needed by IPF is the execution of one or more Lisp forms in parallel with the main process. When the subprocesses are created, the main process does not wait until they are finished but carries on normally with the computation. Hence no communication between processes is necessary. The subprocesses which are schematically represented as arrows labeled *s1* and *s2* in Figure 9.1 are independent of the main process; they may finish before or after the main process.

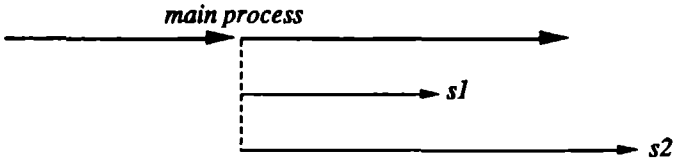


Figure 9.1
Parallel processes are independent of the spawning process

IPF spawns a separate process for each conceptual input. Because the Conceptualizer is not elaborated in the computer model, the main process in the generation system simulates the passing of conceptual messages to the Formulator. This 'passing' is represented as the creation of a grammatical encoding process whose task it is to create part of the syntactic structure for the given conceptual fragment. Meanwhile, the main generation process continues. The separate grammatical encoding processes are schematically represented as arrows labeled *ge1*, *ge2*, and *ge3* in Figure 9.2.

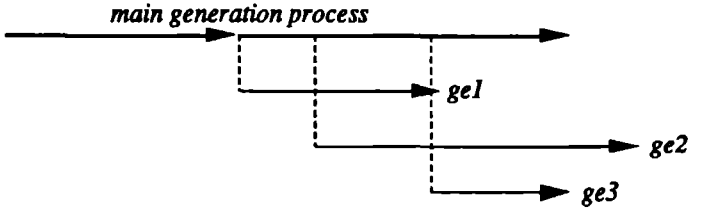


Figure 9.2
Spawning separate grammatical encoding processes during generation

Since IPF employs parallelism, it is worth investigating whether several subtasks of grammatical encoding can also be considered for parallel processing. Without claiming to attain completeness, I believe that several such tasks can be pointed out. In situations involving search, it is often useful to spawn several processes which look for alternatives. The main process waits until at least one of the parallel subprocesses has returned a positive result. This is depicted in the following diagram:

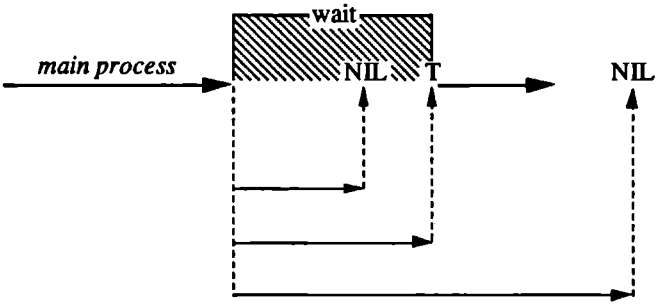


Figure 9.3
The main process continues as soon as one subprocess returns a positive result

One such search occurs during lexicalization (cf. Section 4.2). When more than one possible lexical entry is retrieved from the lexicon, they all try to unify with the current linguistic sign. As soon as one formulation is successful, formulation can continue. This choice could be carried out in parallel. The following expression embodies this choice mechanism:

```
(PARALLEL-MAPCAR-OR #'(LAMBDA (L)
                        (FORMULATE-LEMMA SELF L))
                    LEMMAS)
```

Finally, it is sometimes necessary for a process to wait a while until a condition has been satisfied by some other concurrent process. A timing device, `PROCESS-WAIT-A-WHILE`, allows a process to wait until a condition occurs, but no longer than some predetermined time span. In IPF, such a wait is necessary when a case relation is established between linguistic signs which have not yet been lexicalized. The intercalating segment between such signs cannot be realized until they are lexicalized, because of lexico-syntactic constraints. However, the wait should not extend beyond a certain time span which depends on parameters reflecting constraints of memory and attention.

9.3 Parallel unification in distributed grammatical encoding

When grammatical encoding is distributed among several independent processes, this imposes a processing constraint on the grammar:

> *The grammar of a distributed Grammatical Encoder must allow branches to be added independently of each other and simultaneously.*

Segment Grammar (SG; cf. Chapter 3) in principle allows syntactic structures to be constructed according to this constraint. It will now be shown how grammatical knowledge is *distributed* in SG and how, in this formalism, syntactic tree formation can be modeled as unification distributed among various processors.

In SG, unification is a local operation on nodes in syntactic structures, not on syntactic structures themselves. This locality allows unification to be distributed in the sense that unifications can be carried out simultaneously on different parts of the structure. Parallel unifications are possible because each node in an f-structure contains all necessary features to allow or disallow unification with other nodes.

However, in parallel systems with a shared memory, there may be shared resources which must be protected from other processes when they are used in a critical section. A well-known example of the problem of shared resources in parallelism is the problem of the 'dining philosophers', which is a situation where each philosopher needs both his own and his neighbor's fork in order to eat (Ben-Ari, 1982). Another, more mundane example of a shared resource is that of airplane seats in simultaneous airline reservations. In SG, the shared resources which are to be protected consist of the nodes in the syntactic structure and of the features shared by several nodes. An example of a simultaneous unification of a node with two others is depicted in Figure 9.4.

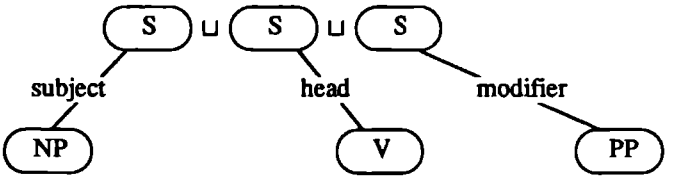


Figure 9.4
Two simultaneous unifications: nodes are shared resources

Even if two unifications involve different nodes, their features can still be shared and hence these features must also be protected during unification. Suppose, e.g., that the unifications in Figure 9.5 happen in parallel. While the features shared by the root and the foot of the S-subject-NP segment are involved in one unification process, the other unification process might change them 'behind its back'.

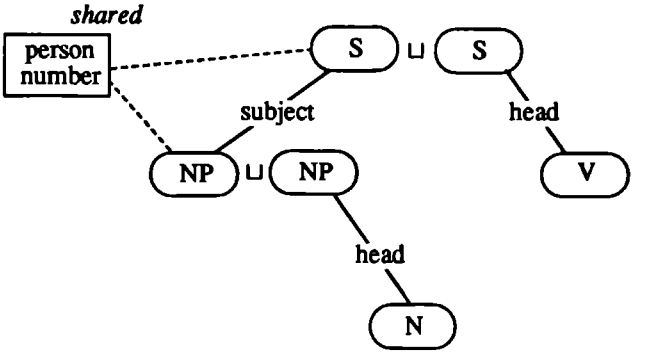


Figure 9.5
Simultaneous unifications: features are shared resources

If we want to allow unification to take place in a distributed fashion, we must take care that the shared resources involved in one unification operation are temporarily made inaccessible to other processes, so that other processes cannot interfere with the delicate checking and changing of features. The simplest solution would be to make unification an 'atomic' operation, which means that other processes cannot run at all during its execution. However, a system based almost completely on unification would then hardly be parallel, since little is performed outside the unifications. A more feasible solution is to *lock* the objects involved in a unification so that they cannot be unified in other processes. A lock is a concurrent programming construct which regulates the access to critical sections of a program. Locks are introduced in SG at the level of unification: the unification mechanism itself is wrapped in a procedure which locks the objects to be unified so that temporarily they cannot be unified in other processes. Naturally, other processes can continue to run; they must wait only if they want to access an object locked by an other process.

In an object-oriented language, a lock can easily be associated with each object. This is implemented as an aspect which contains a list containing a value representing the state of the lock:

```
(DEFOBJECT FEATURE-OBJECT
  (LOCK
    :IF-NEEDED (SELF)
    (LIST NIL))
  ...)
```

Any process wishing to access an object first checks the lock of the object. If it contains a non-NIL value, the object is in a locked state and the process has to wait. If it is NIL, the object is free. The process then inserts its own ID in the lock to signal that the object is busy and proceeds. It is clear that checking the lock and setting it must be performed as one atomic operation, lest another process grabs the lock in between. SCL provides `STORE-CONDITIONAL` as a primitive for atomic conditional assignment. However, unification always involves *two* objects, which must obviously be locked *at the same time*. Therefore, a new atomic operation must be created which can check and set several locks in one atomic operation. This can be implemented with the help of `SCL:WITHOUT-INTERRUPTS`, which disables process scheduling during execution of its body. Now it becomes possible to 'wrap' unification in a procedure `LOCKING-FEATURE-OBJECTS` which first locks two given objects and their features:

```
(DEFOBJECT FEATURE-OBJECT
  (UNIFY
    :FUNCTION (SELF OTHER &KEY (FEATURES (UNION
                                           (FEATURES SELF)
                                           (FEATURES OTHER))))
    (LOCKING-FEATURE-OBJECTS (SELF OTHER)
      (UNIFY-OBJECTS SELF OTHER :FEATURES FEATURES)))
  ...)
```

The Symbolics Lisp machine programming environment offers a utility called PEEK which displays an overview of all processes in the scheduler. Figure 9.6 shows a

snapshot of a PEEK display during a simulation run of IPF. In addition to the main IPF process "Ipf 1", four IPF processes are present, their names starting with "Parallel". Two of those four processes are simultaneously active, as can be seen by their "Run" state. The other two are preparing to perform a unification but must wait for locks to become free; hence their state is "Unification Lock".

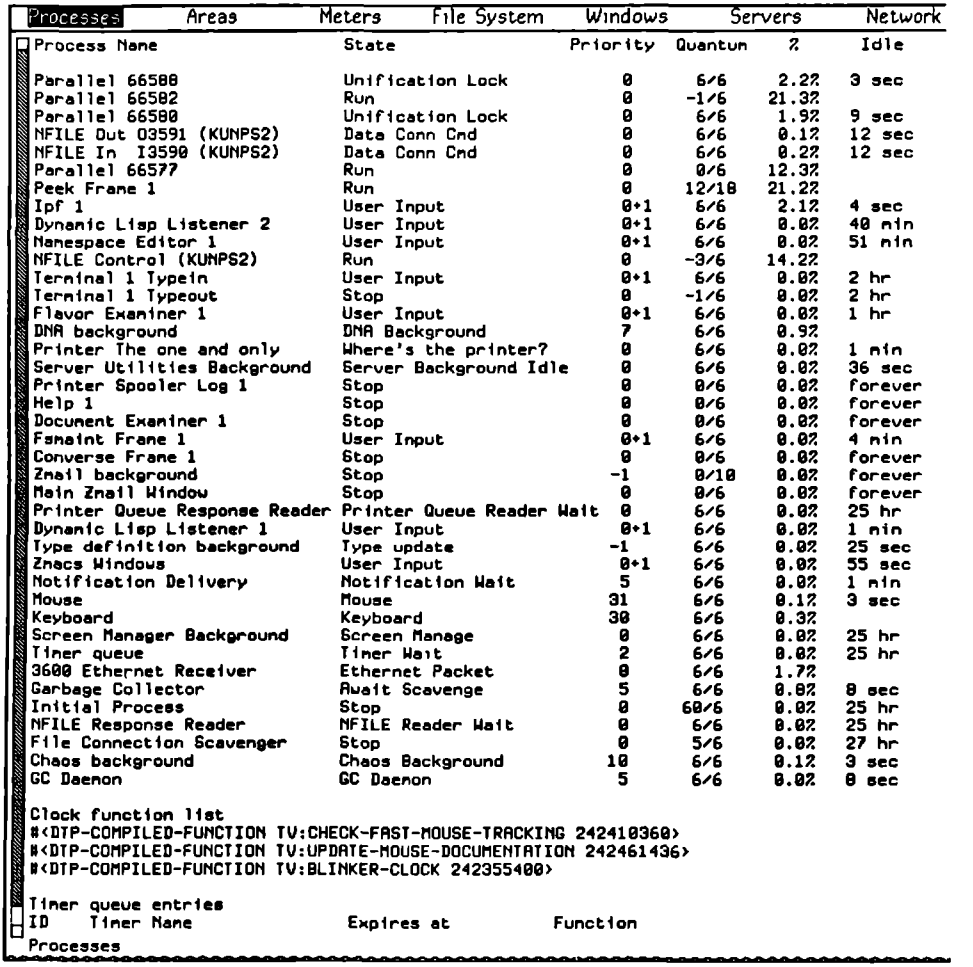


Figure 9.6
A PEEK snapshot of processes in the Lisp machine scheduler during a run of IPF

Notice that many other processes which are unconnected to IPF are also present in the scheduler. Since the scheduler is a finely tuned program, this 'background noise' may influence the scheduler's activities and thus provides a certain randomness which makes the outcome of a simulation run not fully predictable.

9.4 Concluding remarks

Parallelism in generation was suggested by Kempen and Hoenkamp (1987; Hoenkamp, 1983) in their theory of Incremental Procedural Grammar (IPG). However, they did not fully work out how several processes operating in parallel manage the construction of a coherent sentence. This chapter has described a more complete architecture of a Grammatical Encoder which processes conceptual fragments simultaneously and independently of each other. It has also been described how such a system can be implemented in a LISP-based multi-processing environment.

Syntactic knowledge in SG is distributed so that an extension of the formalism which allows parallelism is natural and fairly straightforward. A kind of distributed unification similar to the one presented here seems to be possible in Feature structures based Tree Adjoining Grammar (FTAG), which uses feature unification as a general mechanism to specify restrictions on tree adjoining. Vijay-Shanker and Joshi (1988) claim that FTAG, which allows co-occurrence of features to be stated within elementary trees, can be extended to allow simultaneous adjoining operations in distinct nodes of an elementary tree.

In contrast to the system proposed here, which applies unification locally, other unification-based formalisms, such as FUG, involve arbitrarily large functional descriptions, containing many alternations and arbitrarily distant paths. Although in principle such formalisms could probably be adapted to allow parallel unification, there are to my knowledge no practical systems proposed in the literature.

10 Epilogue

This chapter raises some issues drawn from experience gained with the computer model. IPF and SG are evaluated with respect to their potential for incremental generation. Also, the model is compared to the original IPG model proposed by Kempen and Hoenkamp (1987). Last but not least, some tentative new ideas are proposed in the form of an activation-based extension.

10.1 An evaluation of IPF/SG

This study has described a theory of language which accounts for the fact that in spontaneous speech, speakers construct the syntactic frame of a sentence in a piecemeal fashion. I have focused on both the linguistic formalism (SG) and the process model for grammatical encoding (IPF). The present theory allows small, individual conceptual units to be formulated separately and in parallel, while a general unification mechanism guards consistency in the resulting syntactic structure. Grammatical encoding starts immediately when the first conceptual fragment enters the Formulator. If more fragments are entered, their formulation may overlap in time. Fragments of the f-structure can be composed in upward as well as downward expansions and irrespective of their eventual left-to-right order in the utterance.

10.1.1 A grammar's potential for incremental generation

Several factors can be identified in a grammar which, in general, tend to facilitate an incremental mode of generation:

1 If word order restrictions can be specified so that the one constituent can be positioned quite independently from that of its sister constituents, then it is easier to formulate one constituent at a time. SG realizes this in a practical way by assigning absolute rather than relative positions to constituents.

2 A grammar facilitates incremental generation if it allows variations in word order. There is less load on short term memory if concepts can be uttered roughly as they become accessible. Human speakers take advantage of such ordering possibilities (cf. Pechmann, 1989). The available relative freedom in word order in a language is exploited in IPF by assigning the earliest possible position to a constituent.

3 Incremental generation is less error-prone if a concept can be expressed by various lexical categories. A grammar may allow for such variations by means of productive lexical rules which derive categories from other ones, e.g., nominalization and

passivization. In IPF, such rules are triggered when needed during incremental production.

4 The incremental addition of sister nodes in a syntactic structure is more flexible if constituents are optional. Thus, e.g., an intransitive clause can sometimes be extended to a transitive construction, but remains grammatical without the direct object. Such optional expansions to the structure are possible in SG by the representation of individual ID relationships in syntactic segments.

10.1.2 Comparison with Incremental Procedural Grammar

Kempen and Hoenkamp (1987) present Incremental Procedural Grammar (IPG) as a model of grammatical encoding where linguistic knowledge is represented in a fully procedural way. Although IPF/SG is an incremental procedural grammar in the sense of Kempen and Hoenkamp (1987:209), it separates grammatical knowledge from control knowledge. Categories are represented in IPG as LISP procedures; in SG they are viewed as feature structures (programmed on a computer as objects in CommonORBIT). Whereas ID relationships are represented in IPG as procedure calls, SG renders them as relationships between root and foot in syntactic segments. Whereas in IPG the f-structure is built up as a procedure call hierarchy, SG represents such structures in a declarative way and uses unification as a general structure-building tool.

An important consequence of the separation between linguistic knowledge and control knowledge is that the growth of a syntactic structure in an SG-based model is unbiased with respect to its direction. F-structures can be composed by means of upward as well as downward expansions. In contrast, an IPG procedure call hierarchy can only be extended from the top downward.

Unification in SG is responsible for several tasks which in IPG are based on explicit *rules*. First, the acceptability of categories in ID relationships is decided on by so-called *appointment* rules in IPG; paths of ID relationships at arbitrary distances are specified and put together in a graph of all possible ID relationships (1987:217). SG, in contrast, is careful to keep these relationships separate in a database of segments. This knowledge is exploited by one general mechanism—unification. Second, the addition of function words is handled by so-called *functorization* rules in IPG. Triggered by the presence of features, these rules may add new subprocedure calls. In SG, a set of functor segments can be simply listed in the grammar while letting unification handle the choice between the various possibilities.

Exceptions are handled by special rules in IPG. Kempen and Hoenkamp propose the following procedural mechanisms to handle infinitival complement clauses. Notice the use of procedural terms like *call, var[iable], assign, declare, give:*

(1) a. the V-lemma is stripped of its call to Subj;

b. V is assigned the role to VInfin instead of VFin;

c. the s-var declared by the Obj-S is given a nonstandard value: s-var is initialized not to the procedure's own name but to the value of the s-var that

is within reach (by climbing the procedure call hierarchy).
(Kempen & Hoenkamp, 1987:226)

The information expressed in (1) is specified in SG in a more declarative manner. The segment S-complement-S (cf. Sections 5.3 and 8.3.4) is defined such that the foot is a non-finite S whose address is the address of its mother (thus allowing 'climbing'). Moreover, there are no segments with a non-finite S as their root and the label 'subject' on the arc. Thus nothing needs to be 'stripped' and (1a) is in fact redundant in IPF/SG.

It is not clear how the knowledge in (1) interacts with the rest of the knowledge base in IPG. Presumably, exceptional knowledge such as (1b,c) has a higher priority than default knowledge and somehow blocks default knowledge from applying; other kinds of exceptional knowledge such as (1a) explicitly *remove* default knowledge. In SG, regularities and exceptions are coded in an inheritance network which automatically blocks defaults when an exception is present. E.g., when an S is specified to be non-finite then this will overrule any defaults with respect to this feature (see Chapter 6).

10.2 Activation-based formulation

10.2.1 Speech errors, contextual influences and activation

Although IPF accounts for piecemeal sentence formulation and allows for computational simultaneity, it does not account for *all* phenomena which are often attributed to computational simultaneity. In particular, speech errors (Garrett, 1975) are not directly accounted for. Speech errors are explained nicely by computational simultaneity in *activation spreading* models. In this class of models, the choice between alternative elements to be incorporated in a structure is determined by a number of independent factors which are summed in an *activation* level associated with categories in the lexicon. Activation spreading along links between linguistic units accounts for contextual influences. Thus it is possible to separate a general choice mechanism—which always chooses the element with the highest level of activation—from the various contextual factors determining this activation. The fact that some unintended element is substituted for the intended one is then modeled by an abnormally high activation level of the unintended element due to contextual factors, not by an error in the process control structure. Even without the intent to account for speech errors, activation has also been proposed to sum semantic, syntactic and contextual factors in one network (e.g. Waltz & Pollack, 1985).

IPF currently lacks such a notion of activation, but experience with the model suggests that activation-based processing is compatible with the SG formalism and, perhaps, opens new perspectives for grammatical encoding. Whereas earlier experiments have associated activation only with lexical entries (e.g. Dell, 1986), it seems possible to associate activation with larger units as well. Syntactic categories and syntactic segments (lexical and non-lexical) could very well have an activation level which plays a role in the construction of syntactic structures. In an activation-based version of SG, the activation level of segments determines their chance of being

composed by means of unification. Such a model would account for speech errors with interacting units which are larger than single words as well as for syntactic 'blends'[1].

Contextual influences on grammatical encoding are not only apparent from speech errors, though. Bock (1986) reports *priming* effects on the choice of a syntactic frame. If subjects are primed with certain sentences, then their subsequent descriptions of pictures tend to syntactically congruent with the preceding sentences. This kind of *syntactic persistence* could be modeled by allowing the output of the parsing process to give feedback to the Formulator in the form of activation of category and segment types[2].

Finally, activation-based grammatical encoding could elegantly account for Kolk's (1987) assumptions that aphasia is due to memory limitations rather than to a loss of grammatical knowledge. If, e.g., the activation on syntactic units decays faster than normal, a partial syntactic structure might disintegrate before it is completed.

10.2.2 Backtracking

Does the initiative for generation come from the Conceptualizer or from the Formulator? Given the principle of conceptual guidance, it seems that the initiative should come from the Conceptualizer. A modular theory where information processing components meet the requirement of informational encapsulation (Fodor, 1983) would then assume that there is no direct feedback from the Formulator to the Conceptualizer. However, how are incomplete sentences prevented from being realized? When the Conceptualizer does not know that something is missing—because it does not know about syntactic restrictions—an incomplete utterance could come about: e.g., a subjectless sentence, or a transitive sentence without the obligatory direct object. A solution could consist of a *monitor* which detects errors in the structure built by the Formulator and gives indirect feedback to the Conceptualizer, which then provides an alternative input to the Formulator (Section 1.1). However, this is only useful in so far as the Conceptualizer can provide paraphrases of the original input. But sometimes the alternative can be found within the Formulator itself, e.g., in the form of backtracking and the application of a lexical rule such as passivization; it is then a question of control internal to the Formulator.

The question of backtracking, which could of course also be raised in a non-incremental approach, are nevertheless more pressing in an incremental system where conceptual units are rather small, as in IPF. We have seen that although the successful construction of syntactic structures is in principle independent of the order of conceptual inputs, this cannot always be guaranteed in the present model. Suppose that several concepts are lexicalized independently of one another. Suppose, furthermore,

---

[1] This proposal would treat errors on the level of the f-structure. Levelt (1989:248) suggests that some speech errors can also be interpreted as errors in the derivation of c-structures from f-structures.

[2] Levelt (1989:275) suggests modeling syntactic persistence in IPG as a *bias* in categorial procedures like S and PP (CPROCs). However, it is hard to see how *procedural* knowledge can be biased without introducing a truckload of additional control structure, extra arguments to procedures, etc. The use of activation in elements of a declarative lexicon is a much more general mechanism to achieve the same effect.

that case relations between these concepts are established only after these have been lexicalized. Clearly it is then too late for subcategorization restrictions to guide lexicalization (see Section 4.2). Some kind of backtracking should occur. But backtracking, and undoing previously built structure, is a computationally expensive operation, because it requires a memory of past states of the system. Unifications in SG are about as easily undone as turning a cake back into its ingredients. This is largely because features may have disjunctive values and previous values of a node are lost when the features are unified. A partial solution could consist of developing a version of segment grammar without disjunctive values, but this would of course cause a proliferation of segments.

10.2.3 The 'Unification Space'

For parsing, a new architecture based on Segment Grammar has been proposed by Kempen and Vosse (1989). This parser is based on activation decay and simulated annealing. Its mode of processing is roughly be described as follows:

> "Imagine syntactic tree formation taking place in a 'mental testtube' containing the segments and mobiles retrieved from the lexicon. Assume furthermore that their nodes continuously attempt to combine with other nodes they hit upon, and that the likelihood of a successful combination—or, rather, unification—depends not only on their feature composition but also on their level of activation. Whenever two nodes actually unify, they merge with one another and join together the segments (or trees/mobiles) they belong to, thus effectively creating a larger tree. Unifications ar not granted the life everlasting, though. Depending on the strength of the original bond and due to activation decay, merged nodes may separate again, thereby clearing the way for other—maybe stronger—unifications. This dynamic process of coalescence, disintegration and reintegration will gradually come to rest accordingly as the segments in the testtube succeed in connecting up with one another in bonds of sufficient strength to lend stability to the resulting tree structure." (Kempen & Vosse, 1989:280)

A similarity between backtracking in generation and backtracking in parsing is apparent, because both processes concern the construction of a coherent syntactic structure out of bits and pieces by trying various alternatives. Since the architecture proposed by Kempen and Vosse is based on Segment Grammar, an adaptation to the generation task seems fairly straightforward at first sight. With each segment, an activation level is associated. Rather than simply choosing the first suitable lexical entry in lexicalization, or the first intercalating segment in the intercalation process, all possible segments in the current context are activated and try to combine with the current syntactic structure. The activation level of two nodes is one of the factors defining their probability of getting unified.

While several nodes might initially unify, this structure may disintegrate: as more conceptual input enters the Formulator, another segment which perhaps fits better in the new context may take its place. Thus, the need for backtracking is eliminated in favor of a more general mechanism which is called *annealing*: a steadily dropping

temperature decreases the chance of structures to disintegrate. Finally, a steady equilibrium state is reached with a 'frozen' configuration of segments, which is called a *conformation.*

The 'Unification Space', as this model is called, solves the search problem, accounts for contextual influences, and explains some speech errors. Due to contextual influences or simply the simultaneous presence of equally strong alternatives in the 'unification space', the 'wrong' segments may concatenate or furcate, thus producing speech errors. The unification in Figure 10.1 illustrates how utterance (2a) might come about as a fusion between (2b) and (2c). It is possible that the unification in Figure 10.1 comes about after a period of *oscillation* between various possibilities.

(2) a. * Ik stop eruit

b. Ik stop ermee (I'm quitting)

c. Ik schei eruit (id.)

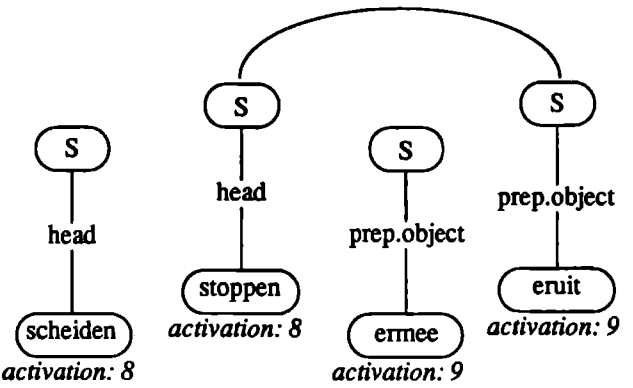


Figure 10.1
Activation-based unification

The new model is compatible with assumptions concerning incremental formulation, parallelism, and the representation of a grammar and lexicon in terms of syntactic segments. However, a number of problems remain unsolved. First, is must be possible to undo unifications as easily as to perform them. A solution might consist of replacing the unification mechanism proposed in Section 8.2 by a kind of 'virtual' unification mechanism which does not destroy the initial configuration of nodes. Second, it is not clear how such a control structure could support two levels of representation—an f-structure and a c-structure—which are both constructed in an incremental fashion. Presumably, for each change to the f-structure, a c-structure must be computed; this could be a computationally very intensive task. Third, it is not straightforward how the output of the unification space might become incrementally available to the next module in the generation process—the Phonological Encoder (see also Section 4.6). If the speaker is to start uttering part of a sentence without immediately being forced to

correct himself, then at least part of the sentence should be in a steady state. A solution may consist of 'freezing' *partial* structures, which signals to the Phonological Encoder that something is ready to be further processed.

A remaining question is, how such a Grammatical Encoder be forced to form just one coherent sentence rather than several isolated parts. As long as annealing results in just *one* conformation, then that conformation is incrementally expanded. When there are two or more conformations, it may be difficult to decide on one of them to be further expanded. The latter may be the case when the conceptualizer has passed *several* concepts to the Formulator but has as yet not established any semantic roles between them. The requirement to arrive at just one conformation can be seen as the principle, that the Grammatical Encoder may construct only one sentence at a time. It is ungrammatical to utter several 'loose' fragments, although they seem to surface sometimes in utterances like (3a). Similarly, a non-retracing repair (or *apokoinou*) e.g. (3b,c) or a dangling participle (3d) could be explained as a temporary instability during annealing which results in two *non-distinct* conformations, i.e., conformations which inadvertently share part of their structure.

(3) a. Susan ... John ... I mean, he decided to marry her.

b. The next speaker will be given by Jonathan Slocum.

c. So we get as an added bonus to this approach is a system which can be extended by the user.

d. Having just returned from New York, my home town looks even smaller than before.

I conclude that activation-based processing is likely an enrichment of the present model, but that further research is needed to test these hypotheses and to smoothly integrate brain-style programming techniques into IPF/SG.

Appendix 1: List of abbreviations

ACL	Association for Computational Linguistics
ADJP	Adjectival Phrase
ADVP	Adverbial Phrase
AI	Artificial Intelligence
Art	Article
Aux	Auxiliary
CG	Categorial Grammar
Comp	Complement
Det	Determiner
Dir.Obj	Direct Object
DPSG	Discontinuous Phrase Structure Grammar
FUG	Functional Unification Grammar
GB	Government-Binding Theory
GPSG	Generalized Phrase Structure Grammar
HPSG	Head-driven Phrase Structure Grammar
ID	Immediate Dominance
IPF	Incremental Parallel Formulator
IPG	Incremental Procedural Grammar
LFG	Lexical-Functional Grammar
LP	Linear Precedence
Mod	Modifier
N	Noun
NP	Noun Phrase
PP	Prepositional Phrase
Prep	Preposition
PS	Phrase Structure
PSG	Phrase Structure Grammar
S	Sentence
SCL	Symbolics Common Lisp
SG	Segment Grammar
Subj	Subject
TGG	Transformational Generative Grammar
V	Verb
VP	Verb Phrase
XOR	logical exclusive disjunction

References

Abeillé, A. & Schabes, Y. 1989. Parsing idioms in lexicalized TAGs. In: *Proceedings of the 4th Conference of the European Chapter of the ACL*, Manchester (pp. 1-9). ACL.

Adriaens, G. 1986. Word Expert Parsing revised and applied to Dutch. In: *Proceedings of the 7th European Conference on Artificial Intelligence* (pp. 222-235).

Allport, A., Antonis, B. & Reynolds, P. 1972. On the division of attention: a disproof of the single channel hypothesis. *Quarterly Journal of Experimental Psychology* 24, 225-235.

André, E., Herzog, G. & Rist, Th. 1988. On the simultaneous interpretation of real world image sequences and their natural language description: the system SOCCER. In: Kodratoff, Y. (ed.) *Proceedings of the 8th European Conference on Artificial Intelligence* (pp. 449-454). London: Pitman.

André, E., Herzog, G. & Rist, Th. 1989. *Natural language access to visual data: dealing with space and movement*. Report No. 63, SFB 314 (VITRA), University of Saarbrücken.

Appelt, D. 1983. TELEGRAM: a grammar formalism for language planning. In: *Proceedings of the 8th IJCAI*, Karlsruhe.

Appelt, D. 1985. *Planning English sentences*. Cambridge: Cambridge University Press.

Ben-Ari, M. 1982. *Principles of concurrent programming*. Englewood Cliffs: Prentice-Hall.

Bobrow, D.G. & Winograd, T. 1977. An overview of KRL, a Knowledge Representation Language. *Cognitive Science* 1, 3-46.

Bobrow, R.J. & Webber, B.L. 1980. Knowledge representation for syntactic/semantic processing. In: *Proceedings of the First Annual National Conference on Artificial Intelligence* (pp. 316-323). Stanford University.

Bock, J.K. & Warren, R. 1985. Conceptual accessibility and syntactic structure in sentence formulation. *Cognition* 21, 47-67.

Bock, J.K. 1986. Syntactic persistence in language production. *Cognitive Psychology* 18, 355-387.

Bock, J.K. 1987a. Exploring levels of processing in sentence production. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 351-363). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

Bock, J.K. 1987b. An effect of the accessibility of word forms on sentence structures. *Journal of memory and language* 26, 119-137.

Boomer, D. 1965. Hesitation and grammatical encoding. *Language and Speech* 8, 148-158.

Bos, E., Claassen, W. & Huls, C. (forthcoming) The Pooh way in human-computer interaction: a multimodal user interface. To appear in: *Proceedings of the 5th European Conference on Cognitive Ergonomics (ECCE-5)*, Siena, Italy, 3-6 September 1990.

Bouma, G. 1986. Kategoriale Grammatica en het Warlpiri. *Glot* 8, 227-256.

Bouma, G. 1989. Efficient processing of flexible categorial grammar. In: *Proceedings of the 4th Conference of the European Chapter of the ACL*, Manchester (pp. 19-26). ACL.

Brachman, R.J. & Schmolze, J.G. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9, 171-216.

Brachman, R.J. 1978. *A structural paradigm for representing knowledge*. Report 3605, Bolt, Beranek and Newman Inc., Cambridge, MA.

Brachman, R.J. 1979. On the epistemological status of semantic networks. In: Findler, N.V. (ed.) *Associative networks: representation and use of knowledge by computers* (pp. 3-50). New York: Academic Press.

Bresnan, J. (ed.) 1982. *The mental representation of grammatical relations*. Cambridge, MA: MIT Press.

Bresnan, J. 1978. A realistic transformational grammar. In: Halle, M., Bresnan, J. & Miller, G. (eds.) Linguistic theory and psychological reality. Cambridge, MA: MIT Press.

Bresnan, J., Kaplan, R.M., Peters, S., & Zaenen, A. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry* 13, 613-635.

Brewka, G. 1989. Nonmonotonic logics—a brief overview. *AI Communications* 2, 88-97.

Brotherton, P.L. 1979. Speaking and not speaking: processes for translating ideas into speech. In: Siegman, A.W. & Feldstein, S. (eds.) *Of speech and time: temporal speech patterns in interpersonal contexts*. Hillsdale, NJ: Lawrence Erlbaum.

Bunt, H.C. 1988. DPSG and its use in sentence generation from meaning representations. In: Zock, M. & Sabah, G. (eds.) *Advances in natural language generation, Vol 2* (pp. 1-26). London: Pinter Publishers.

Calder, J. 1989. Paradigmatic morphology. In: *Proceedings of the 4th Conference of the European Chapter of the ACL*, Manchester (pp. 58-65). ACL.

Chomsky, N. 1959. Review of Skinner (1957). *Language* 35, 26-58.

Chomsky, N. 1965. *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.

Chomsky, N. 1970. Remarks on nominalization. In: Jacobs, R. & Rosenbaum, P. (eds.) *Readings in English transformational grammar*. Waltham, Mass.: Blaisdell.

Chomsky, N. 1981. *Lectures on government and binding*. Dordrecht: Foris.

Daelemans, W. 1987a. *Studies in language technology: an object-oriented model of morphophonological aspects of Dutch*. Ph.D. dissertation, University of Leuven, Department of Linguistics.

Daelemans, W. 1987b. A tool for the automatic creation, extension and updating of lexical knowledge bases. In: *Proceedings of the 3rd Conference of the European Chapter of the ACL*, Copenhagen (pp. 70-74). ACL.

Daelemans, W. 1988a. A model of Dutch morphophonology and its applications. *AI Communications* 1 (2), 18-25.

Daelemans, W. 1988b. GRAFON: a grapheme-to-phoneme conversion system for Dutch. In: *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest (pp. 133-138).

Daelemans, W. 1989. Object-oriented hierarchical lexicons. In: *Proceedings of the IJCAI Workshop on Object-Oriented Programming in AI*, Detroit.

De Schutter, G. 1976. *De bouw van de Nederlandse zin: beschrijving en voorstel tot beregeling*. Verslagen en Mededelingen van de Koninklijke Academie voor Nederlandse Taal- en Letterkunde, nr. 2.

De Smedt, K. & Kempen, G. 1987. Incremental sentence production, self-correction and coordination. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 365-376). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

De Smedt, K. & Kempen, G. 1990. Discontinuous constituency in Segment Grammar. In: *Proceedings of the Symposium on Discontinuous Constituency*, Tilburg (pp. 97-111).

De Smedt, K. 1980. *Using case to improve frame-based representation languages*. Schlumberger A.I. Working paper No. 3. Ridgefield, Conn.: Schlumberger-Doll Research.

De Smedt, K. 1984. Using object-oriented knowledge-representation techniques in morphology and syntax programming. In: O'Shea, T. (ed.) *Proceedings of the 6th European Conference on Artificial Intelligence* (pp. 181-184). Amsterdam: Elsevier.

De Smedt, K. 1987. Object-oriented programming in Flavors and CommonORBIT. In: Hawley, R. (ed.) *Artificial Intelligence programming environments* (pp. 157-176). Chichester: Ellis Horwood.

De Smedt, K. 1989. *Object-oriented knowledge representation in CommonORBIT*. Internal Report 89-NICI-01, University of Nijmegen, Nijmegen Institute for Cognition research and Information technology (NICI).

De Smedt, K., Geurts B. & Desain P. 1987. Waiting for the gift of sound and vision. In: *Proceedings of the ESPRIT Natural Language and Dialogue Workshop*, Brussels, 1 October 1987.

Dekeyser, X., Devriendt, B., Tops, G.A.J., Geukens, S. 1979. *Foundations of English grammar*. Antwerpen: De Nederlandsche Boekhandel.

Dell, G.S. 1986. A spreading-activation theory of retrieval in sentence production. *Psychological Review* 93, 238-321.

Dik, S. 1978. *Functional Grammar*. Amsterdam: North Holland.

Ducournau, R. & Habib, M. 1987. On some algorithms for multiple inheritance in object-oriented programming. In: *Proceedings of ECOOP'78 (Bigre+Globule 54)*, 291-300. Paris: AFCET.

Elman, J.L. 1989. Structured representations and connectionist models. In: *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, Ann Arbor (pp. 17-25). Hillsdale: Lawrence Erlbaum.

Evans, R. & Gazdar, G. 1989. Inference in DATR. In: *Proceedings of the 4th Conference of the European Chapter of the ACL*, Manchester (pp. 66-71). ACL.

Ferber, J. & Volle, Ph. 1988. Using coreference in object-oriented representations. In: Kodratoff, Y. (ed.) *Proceedings of the 8th European Conference on Artificial Intelligence* (pp. 238-240). London: Pitman.

Fikes, R. & Kehler, T. 1985. The role of frame-based representation in reasoning. *CACM* 28, 904-920.

Fillmore, Ch. 1968. The case for case. In: Bach, E. & Harms, R.T. (eds.) *Universals in linguistic theory* (pp. 1-88). New York: Holt, Rinehart & Winston.

Fillmore, Ch. 1977. The case for case reopened. In: Cole, P. & Sadock, J.M. (eds.) *Syntax and Semantics. Vol. 8: Grammatical Relations* (pp. 59-81). New York:

Flickinger, D., Pollard, C. & Wasow, Th. 1985. Structure-sharing in lexical representation. In: *Proceedings of the 23rd Annual Meeting of the ACL* (pp. 262-267). ACL.

Florijn, A. (in preparation) *De beregeling van woordvolgorde in het Nederlands*. Ph.D. dissertation, University of Amsterdam.

Fodor, J.A. 1983. *The modularity of mind*. Cambridge, MA: MIT Press.

Ford, M. & Holmes, V.M. 1978. Planning units and syntax in sentence production. *Cognition* 6, 35-53.

Frijda, N.H. 1972. Simulation of human long-term memory. *Psychological Bulletin* 77, 1-31.

Garrett, M. 1975. The analysis of sentence production. In: Bower, G. (ed.) *The psychology of learning and motivation (Vol. 9)*. New York: Academic Press.

Garrett, M. 1980. Levels of processing in sentence production. In: Butterworth, B. (ed.) *Language production. Vol. 1: Speech and Talk*. New York: Academic Press.

Gazdar, G. 1982. Phrase structure grammar. In: Jacobson, P. & Pullum, G. K. (eds.), *The Nature of Syntactic Representation* (pp. 131-186). Dordrecht: Reidel.

Gazdar, G. 1987. Linguistic applications of default inheritance mechanisms. In: Whitelock, P., Wood, M., Somers, H., Johnson, R., & Bennett, P. (eds.) *Linguistic theory and computer applications* (pp. 37-67). London: Academic Press.

Gazdar, G., Klein, E., Pullum, G. & Sag, I. 1985. *Generalized Phrase Structure Grammar*. Oxford: Basil Blackwell.

Goldman-Eisler, F. 1972. Pauses, clauses, sentences. *Language and Speech* 15, 103-113.

Halle, M. 1973. Prolegomena to a theory of word formation. *Linguistic Inquiry* 4(1), 3-16.

Herzog, G., Sung, C.-K., André, E., Enkelmann, W., Nagel, H.-H., Rist, T., Wahlster, W. & Zimmermann, G. 1989. *Incremental natural language description of dynamic imagery*. Report 58, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken.

Hetzron, R. 1975. Where the grammar fails. *Language* 51, 859-872.

Hoekstra, T., Van der Hulst, H. & Moortgat, M. (eds.) 1980. *Lexical grammar*. Dordrecht: Foris.

Hoenkamp, E. 1980. Spontaneous speech as a feedback process. In: *AISB-80. Proceedings of the AISB Conference on Artificial Intelligence*, Amsterdam.

Hoenkamp, E. 1983. *Een computermodel van de spreker: psychologische en linguistische aspecten*. Ph.D. dissertation, University of Nijmegen.

Horn, G.M. 1985. Raising and complementation. *Linguistics* 23, 813-850.

Hudson, R. 1984. *Word Grammar*. Oxford: Basil Blackwell.

Jackendoff, R. 1975. Morphological and semantic regularities in the lexicon. *Language* 51, 639-671.

Jacobs, P.S. 1985. *A knowledge-based approach to language production*. Report No. UCB/CSD 86/254. Ph.D. dissertation, University of California, Berkeley.

Jacobs, P.S. 1987. KING: a knowledge-intensive natural language generator. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 219-230). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

Joshi, A. 1987. The relevance of tree adjoining grammar to generation. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 233-252). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

Kaplan, R. & Bresnan, J. 1982. Lexical-functional grammar: A formal system for grammatical representation. In: Bresnan, J. (ed.) *The mental representation of grammatical relations*. Cambridge, MA: MIT Press.

Karttunen, L. 1984. Features and values. In: *Proceedings of the 10th International Conference on Computational Linguistics*, Stanford (pp. 28-33). ACL.

Kay, M. 1979. Functional grammar. In: *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society* (pp. 142-158). Berkeley Linguistic Society.

Kay, M. 1985. Parsing in functional unification grammar. In: Dowty, D.R., Karttunen, L. & Zwicky, A.M. (eds.) *Natural language parsing: psychological, computational and theoretical perspectives* (Chapter 7, pp. 251-278). Cambridge: Cambridge University Press.

Keenan, E. & Comrie, B. 1977. Noun phrase accessibility and universal grammar. *Linguistic Inquiry* 8, 63-99.

Keenan, E. & Hawkins, S. 1987. The psychological validity of the accessibility hierarchy. In: E. Keenan, *Universal Grammar: 15 Essays*. London: Croom Helm.

Kempen, G. & Hoenkamp, E. 1987. An incremental procedural grammar for sentence formulation. *Cognitive Science* 11, 201-258.

Kempen, G. & Huijbers, P. 1983. The lexicalization process in sentence production and naming: indirect election of words. *Cognition* 14, 185-209.

Kempen, G. & Vosse, Th. 1989. Incremental syntactic tree formation in human sentence processing: a cognitive architecture based on activation decay and simulation annealing. *Connection Science* 1, 275-292.

Kempen, G. 1978. Sentence construction by a psychologically plausible formulator. In: Campbell, R. & Smith, P. (eds.) *Recent advances in the psychology of language (Vol. 2: formal and experimental approaches)*. New York: Plenum Press.

Kempen, G. 1987. A framework for incremental syntactic tree formation. In: *Proceedings of the 10th IJCAI*, Milan (pp. 655-660). Los Altos: Morgan Kaufmann.

Kitano, Hiroaki. 1989. A massively parallel model of natural language generation for interpreting telephony: almost concurrent processing of parsing and generation. In: *Extended abstracts presented at the Second European Natural Language Generation Workshop*, Edinburgh. University of Edinburgh.

Kolk, H. 1987. A theory of grammatical impairment in aphasia. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 377-391). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

Kukich, K. 1987. Where do phrases come from: some preliminary experiments in connectionist phrase generation. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 405-421). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

Lakoff, G. 1987. *Women, fire, and dangerous things*. Chicago: University of Chicago Press.

Levelt, W. 1983. Monitoring and self-repair in speech. *Cognition* 14, 41-104.

Levelt, W. 1989. *Speaking: from intention to articulation*. Cambridge, MA: MIT Press.

Lieberman, H. 1986. Using prototypical objects to represent shared behavior in object-oriented systems. In: *Proceedings of the First ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, Portland (Oregon). *SigPlan Notices* 21, 214-223.

McDonald, D. & Pustejovsky, J. 1985a. A computational theory of prose style for natural language generation. In: *Proceedings of the 2nd Conference of the European Chapter of the ACL*, Geneva. ACL.

McDonald, D. & Pustejovsky, J. 1985b. TAGs as a grammatical formalism for generation. In: *Proceedings of the 23rd Annual Meeting of the ACL*, Chicago. ACL.

Minsky, M. 1975. A framework for representing knowledge. In: Winston, P. (ed.) *The psychology of computer vision* (pp. 211-277). New York: McGraw-Hill.

Newell, A. 1981. The knowledge level. *AI Magazine*, Summer 1981, 1-20.

Paul, H. 1909. *Prinzipien der Sprachgeschichte*. (14th ed.) Halle a.S.: Niemeyer.

Pechmann, Th. 1987. *Effects of incremental speech production on the syntax and content of noun phrases*. Report 20, Arbeiten der Fachrichtung Psychologie, Universität des Saarlandes, Saarbrücken.

Pechmann, Th. 1989. Incremental speech production and referential overspecification. *Linguistics* 27, 89-110.

Petrie, H. 1989. Syntactic planning in human speech production. In: *Extended abstracts presented at the Second European Natural Language Generation Workshop, Edinburgh*. University of Edinburgh.

Phillips, B. 1984. An object-oriented parser. In: Bara, B.G. & Guida, G. (eds.) *Computational models of natural language processing* (pp. 297-321). Amsterdam: North-Holland.

Pollard, C. & Sag, I. 1987. *Information-based syntax and semantics*. Stanford: CSLI.

Proudian, D. & Pollard, C. 1985. Parsing head-driven phrase structure grammar. In: *Proceedings of the 23rd Annual Meeting of the ACL* (pp. 167-171). ACL.

Reithinger, N. 1987. Generating referring expressions and pointing gestures. In: Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics* (pp. 71-81). Dordrecht/Boston/Lancaster: Kluwer Academic Publishers.

Sag, I. & Pollard, C. 1987. *Head-driven phrase structure grammar: an informal synopsis*. Report CSLI-87-79, Center for the Study of Language and Information. Stanford University.

Schabes, Y., Abeillé, A. & Joshi, A.K. 1988. Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In: *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest.

Schotel, H. 1987. *Programmeren in PROLOG*. Muiderberg: Coutinho.

Schriefers, H. & Pechmann, Th. 1988. Incremental production of referential noun-phrases by human speakers. In: Zock, M. & Sabah, G. (eds.) *Advances in natural language generation, Vol. 1* (pp. 172-179). London: Pinter Publishers.

Sells, P. 1985. *Lectures on contemporary syntactic theories*. CSLI Lecture notes Nr. 3. Stanford: CSLI.

Shieber, S.M. 1986. *An introduction to unification-based approaches to grammar*. CSLI Lecture Notes 4. Stanford: CSLI.

Skinner, B.F. 1957. *Verbal behavior*. New York: Appleton-Century-Crofts.

Small, S. 1983. Parsing as cooperative distributed inference: Understanding through memory interactions. In: King, M. (ed.) *Parsing natural language* (pp. 245-276). London: Academic Press.

Stanley, R. 1967. Redundancy rules in phonology. *Language* 43(1).

Starosta, S. 1978. The one per sent solution. In: Abraham, W. (ed.) *Valence, semantic case and grammatical relations* (pp. 459-576). Amsterdam.

Steele, G.L. 1984. *Common LISP: the language*. Digital Press.

Steels, L. & De Smedt, K. 1983. Some examples of frame-based syntactic processing. In: Daems, Fr. & Goossens, L. (eds.) *Een spyeghel voor G. Jo Steenbergen* (pp. 293-305). Leuven: Acco.

Steels, L. 1978a. *Frame-based knowledge representation*. Working Paper 170, MIT AI Laboratory, Cambridge, MA.

Steels, L. 1978b. *Introducing Conceptual Grammar*. Working Paper 176, MIT AI Laboratory, Cambridge, MA.

Steels, L. 1978c. *Some examples of Conceptual Grammar*. Working Paper 177, MIT AI Laboratory, Cambridge, MA.

Steels, L. 1979. *Reasoning modeled as a society of communicating experts*. Technical Report 524, MIT AI Laboratory, Cambridge, MA.

Steels, L. 1983. ORBIT: An applicative view of object-oriented programming. In: Degano & Sandewall (eds.), *Proceedings of the European Conference On Integrated Interactive Computing Systems*, Stresa. Amsterdam: North-Holland.

Tesnière, L. 1959. *Eléments de syntaxe structurale*. Paris: Klincksieck.

Touretzky, D.S., Horty, J.F. & Thomason, R.H. 1987. A clash of intuitions: The current state of nonmonotonic multiple inheritance systems. In: *Proceedings of the 10th IJCAI*, Milan. Los Altos: Morgan Kaufmann.

Van der Linden, E., Brinkkemper, S., De Smedt, K., Van Boven, P., and Van der Linden, M. 1989. The representation of lexical objects. In: Magay, T. & Zigány, J. (eds.) *Proceedings of the EURALEX Third International Congress*, Budapest, 4-9 September 1988. Budapest: Akadémiai Kiaido. (Also published as Internal Report 88-ITI-B-33, TNO, Delft).

Van Wijk, C. & Kempen, G. 1987. A dual system for producing self-repairs in spontaneous speech: evidence from experimentally elicited corrections. *Cognitive Psychology* 19, 403-440.

Vijay-Shankar, K. & Joshi, A.K. 1985. Some computational properties of Tree Adjoining Grammars. In: *Proceedings of the 23rd Annual Meeting of the ACL*, Chicago. ACL.

Vijay-Shanker, K. & Joshi, A.K. 1988. Feature structures based Tree Adjoining Grammars. In: *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest. ACL.

Von Kleist, H. 1805. Über die allmähliche Verfertigung der Gedanken beim Reden. In: Sembdner, H. (ed.) *Sämtliche Werke und Briefe*. Darmstadt: Wissenschaftliche Buchgesellschaft.

Waltz, D.L. & Pollack, J.B. 1985. Massively parallel parsing: a strongly interactive model of natural language interpretation. *Cognitive Science* 9, 51-74.

Wedekind, J. 1988. Generation as structure driven derivation. In: *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest (pp. 732-737). ACL.

Weinreb, D. & Moon, D. 1980. *Flavors: message passing in the Lisp Machine*. Memo AIM-602, MIT, Cambridge, MA.

Wirth, N. 1971. Program development by stepwise refinement. *Communications of the ACM* 14, 221-227.

Samenvatting

Spontaan spreken wordt gekenmerkt door het feit, dat de spreker soms de inhoud van de zin nog niet volledig heeft bepaald terwijl de eerste woorden al worden uitgesproken. Het stukje bij beetje (incrementeel) bedenken en uitspreken van een zin betekent dat ook de syntactische structuur van een zin stukje bij beetje wordt opgebouwd. Een model voor zinsgeneratie moet daarom bestaan uit modulen die *parallel* kunnen werken: terwijl de spreker een deel uitspreekt, bedenkt hij verdere inhouden en bouwt hij de syntactische structuur van de zin verder uit. De belangrijkste modulen die hierbij betrokken zijn noemen we de Conceptualisator, de Formulator en de Articulator (Hoofdstuk 1).

Deel Een bespreekt psychologische en linguïstische aspecten van het computermodel IPF (Incrementele Parallelle Formulator). Het is maar ten dele bekend in welke mate het generatieproces incrementeel verloopt; tevens zijn hiervoor nog geen goede simulatiemodellen ontwikkeld. Wel kan men mogelijke toepassingen aanduiden, o.a. simultaanvertaling, multimodale interfaces en simultane interpretatie en beschrijving van bewegende beelden (Hoofdstuk 2). Voor incrementele zinsgeneratie lijkt het noodzakelijk om niet alleen volledige zinnen, maar ook correcte zins*delen* en zelfs losse onderdelen van woordgroepen te kunnen aanmaken. Tevens moet het mogelijk zijn om deze partiële structuren verder uit te breiden met minimale herzieningen. Daarvoor wordt *Segmentgrammatica* voorgesteld, een formalisme dat gebaseerd is op *syntactische segmenten* die ieder een individuele syntactische relatie voorstellen. Door deze minimale elementen met elkaar te verbinden door middel van *unificatie* kan een zinsstructuur stukje bij beetje worden opgebouwd (Hoofdstuk 3). Het zinsbouwproces hoeft overigens niet in een bepaalde volgorde te verlopen. Als verschillende delen van de inhoud van een te construeren zin min of meer tegelijk worden aangeboden aan de Formulator, dan kan deze module verschillende delen van de syntactische boom tegelijk aanmaken. Het zinsbouwproces in de Formulator is zo georganiseerd, dat in feite twee structuren worden gebouwd: een functionele structuur (*f-structuur*), waarin syntactische relaties worden gerepresenteerd, en een constituentenstructuur (*c-structuur*), waarin de groepering en volgorde van woorden wordt gerepresenteerd. Omdat zinsstructuren stukje bij beetje worden opgebouwd naargelang er invoer komt van de Conceptualisator, kan het verloop van die invoer in de tijd mede de vorm en volgorde van de zinsdelen bepalen. Daardoor kunnen variaties in woordvolgorde in zekere mate worden verklaard, bijvoorbeeld het verschil tussen "Jan ... speelde vorige week in Amsterdam" en "Vorige week ... speelde Jan in Amsterdam" (Hoofdstuk 4). Vanuit taalkundig oogpunt kan de verhouding tussen f-structuur en c-structuur in

Segmentgrammatica een belangrijk aantal *discontinuïteiten* in zinnen weergeven, bijvoorbeeld extrapositie in "Ik heb *een auto* gekocht ... *met zes deuren*" (Hoofdstuk 5).

Deel Twee bespreekt aspecten van representatie en computerprogrammering in IPF. Omdat zinsgeneratie een zeer kennisintensief proces is, moet worden gestreefd naar een representatie van linguïstische kennis die redundantie vermijdt; een raamwerk voor kennisrepresentatie moet daarom voorzien in mechanismen voor *generalisatie* en *specialisatie*. In een hiërarchische structurering van taalkundige concepten kan een concept model staan voor een ander. Op die manier kan een concept kennis *erven* van een *prototype* zonder die kennis te hoeven dupliceren. Het netwerk van overervingsrelaties tussen linguïstische categorieën vormt een hiërarchie waarin taalkennis efficiënt wordt gerepresenteerd. Tevens blijft het mogelijk om uitzonderingen op te nemen zonder dat dit invloed heeft op de geldigheid van algemenere kennis. Een *object-gerichte* programmeertaal, bijvoorbeeld de op LISP gebaseerde taal CommonORBIT, bevat de nodige constructies voor het opbouwen van zo'n hiërarchie van taalkundige *objecten* (Hoofdstukken 6 en 7). De taalkundige eenheden in Segmentgrammatica kunnen goed worden gerepresenteerd in CommonORBIT, dat tevens voorziet in een mechanisme voor *unificatie* van objecten (Hoofdstuk 8). Wanneer unificaties echter in onafhankelijke, parallelle processen verlopen, dan is het niet vanzelfsprekend dat die ene zinsstructuur waarop zij werken wel consistent blijft. IPF wendt technieken van *concurrent* programmeren aan voor het distribueren van de zinsbouw over aparte parallelle processen (Hoofdstuk 9). Het IPF-model kan worden gezien als een uitbreiding van IPG (Kempen & Hoenkamp, 1987) maar verklaart nog niet alles met betrekking tot incrementele zinsgeneratie. Met name versprekingen en afasie eisen een architectuur waarin de tijd een nog belangrijker rol speelt bij de interactie van linguïstische eenheden. Een uitbreiding gebaseerd op activatieverval kan daarom een nog adequater model opleveren (Hoofdstuk 10).

Curriculum vitae

Koenraad De Smedt werd geboren te Wilrijk (België) op 30 oktober 1954. Hij studeerde Germaanse Filologie aan de Universitaire Faculteiten Sint-Ignatius Antwerpen (UFSIA) en de Universitaire Instelling Antwerpen (UIA). Na het behalen van zijn licentiaatsdiploma in 1977 en het verrichten van zijn vervangende dienstplicht als gewetensbezwaarde werd hij aangesteld als aspirant wetenschappelijk navorser bij het Belgisch Nationaal Fonds voor Wetenschappelijk Onderzoek (NFWO). In deze functie, die hij aan de UIA uitoefende, begon hij in 1979 een onderzoek naar de objectgerichte representatie van taalkundige kennis. In 1983 trad hij in dienst bij de Katholieke Universiteit Nijmegen (KUN) als wetenschappelijk onderzoeker. Hij was aanvankelijk werkzaam bij het Taaltechnologieproject, dat beoogde om enkele praktische toepassingen op het gebied van de computertaalkunde te ontwikkelen. Sinds oktober 1985 is hij universitair docent aan de KUN in de studierichtingen Cognitiewetenschap en Psychologie. Hij verricht tevens onderzoek op het gebied van de computationele psycholinguïstiek bij het Nijmeegs Instituut voor Cognitie-onderzoek en Informatietechnologie (NICI).